#WWDC18

Using Collections Effectively

Session 229

Michael LeHew, Foundation

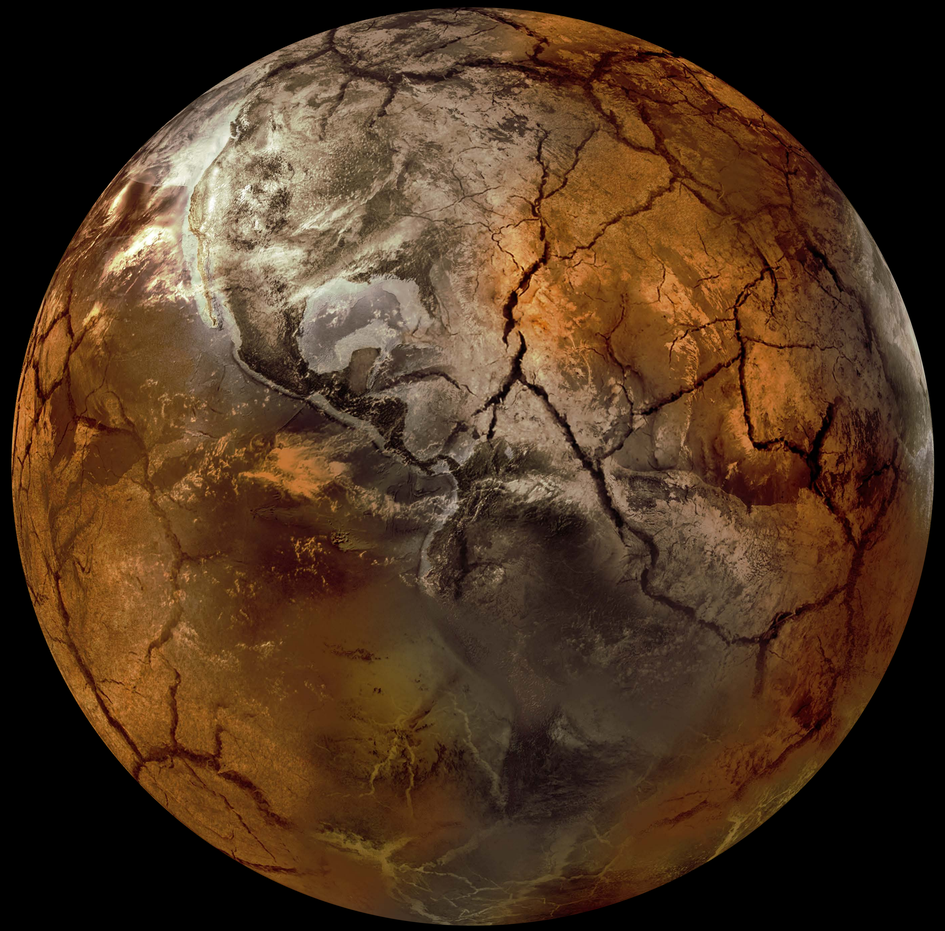
Fundamentals

Indices and Slices

Lazy

Mutability and Multithreading

Foundation and Bridging



```
// A World Without Arrays
```

```
// A World Without Arrays

let bear1 = "Grizzly"
```

```
// A World Without Arrays

let bear1 = "Grizzly"
let bear2 = "Panda"
let bear3 = "Polar"
let bear4 = "Spectacled"
```

```
// A World Without Arrays

let bear1 = "Grizzly"
let bear2 = "Panda"
let bear3 = "Polar"
let bear4 = "Spectacled"

print("\(bear1) bear") // Grizzly bear
print("\(bear2) bear") // Panda bear
print("\(bear3) bear") // Polar bear
print("\(bear4) bear") // Spectacled bear
```

```
// A World Without Dictionaries

func habitat(for bear: String) -> String? {









}
```

```
// A World Without Dictionaries

func habitat(for bear: String) -> String? {
    if bear == "Polar" {
        return "Arctic"
    } else if bear == "Grizzly" {
        return "Forest"
    } else if bear == "Brown" {
        return "Forest"
    } else if /* all the other bears */
    ...
    return nil
}
```

```
// A World Without Dictionaries

func habitat(for bear: String) -> String? {
   if bear == "Polar" {
      return "Arctic"
   } else if bear == "Grizzly" {
      return "Forest"
   } else if bear == "Brown" {
      return "Forest"
   } else if /* all the other cases */
   ...
   return nil
}

let bear = "Polar"
print("\(bear) bears live in the \(habitat(for: bear).")
```

```
// Our World Has Collections

let bear = ["Grizzly", "Panda", "Polar", "Spectacled"]
let habitats = ["Grizzly": "Forest", "Polar": "Arctic"]
```

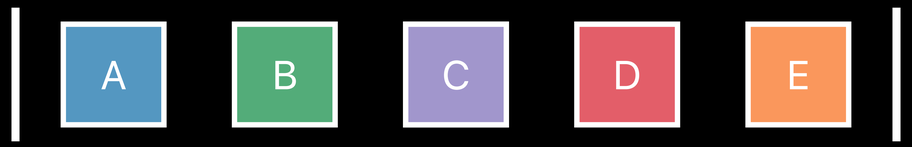
```swift
// Our World Has Collections

let bear = ["Grizzly", "Panda", "Polar", "Spectacled"]
let habitats = ["Grizzly": "Forest", "Polar": "Arctic"]

for bear in bears {
  print("\(bear) Bear")
}
```

```
// Our World Has Collections

let bear = ["Grizzly", "Panda", "Polar", "Spectacled"]
let habitats = ["Grizzly": "Forest", "Polar": "Arctic"]

for bear in bears {
  print("\(bear) Bear")
}

let bear = bears[2]
let habitat = habitats[bear] ?? ""
print("\(bear) bears live in the \(habitat)")
```

```
protocol Collection
```

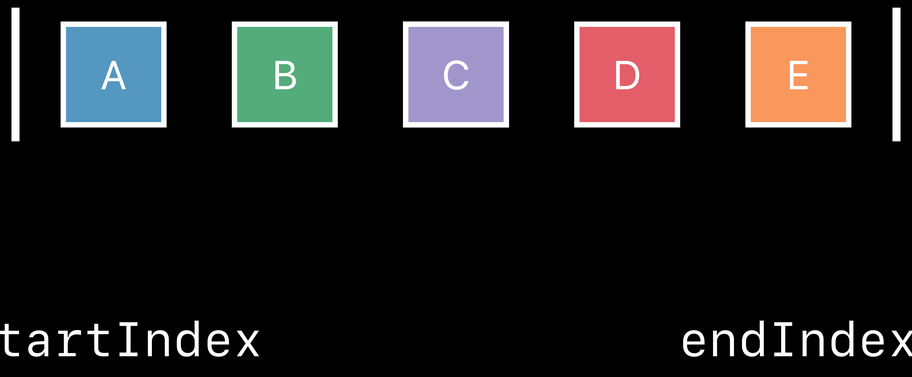
Collections Store Elements

Collections Store Elements

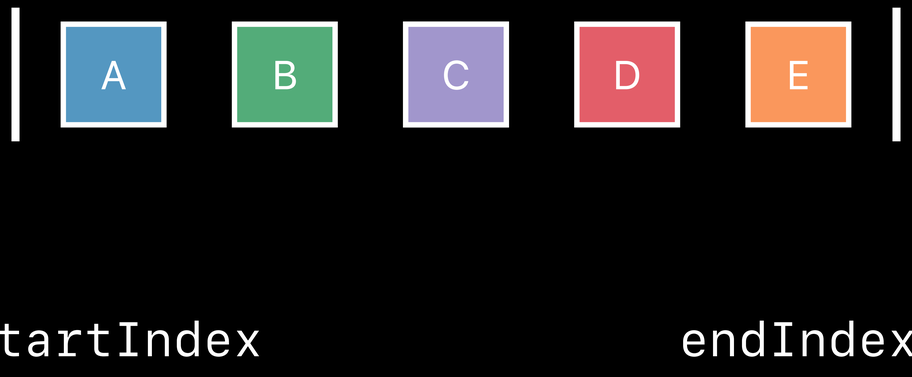


`startIndex`

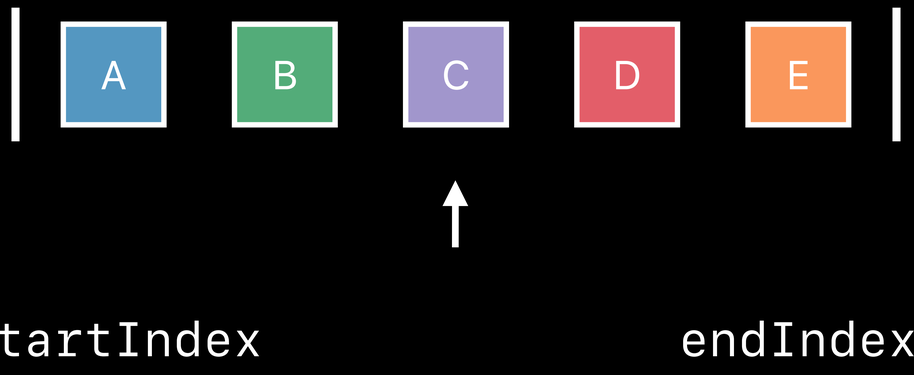
Collections Store Elements



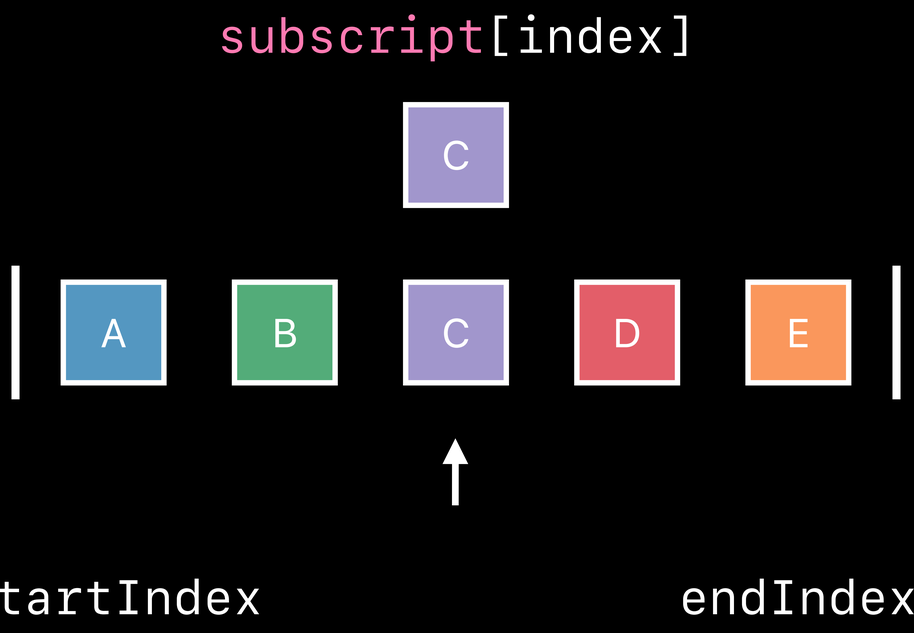
Collections Store Elements



Collections Store Elements



Collections Store Elements



```
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

```
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

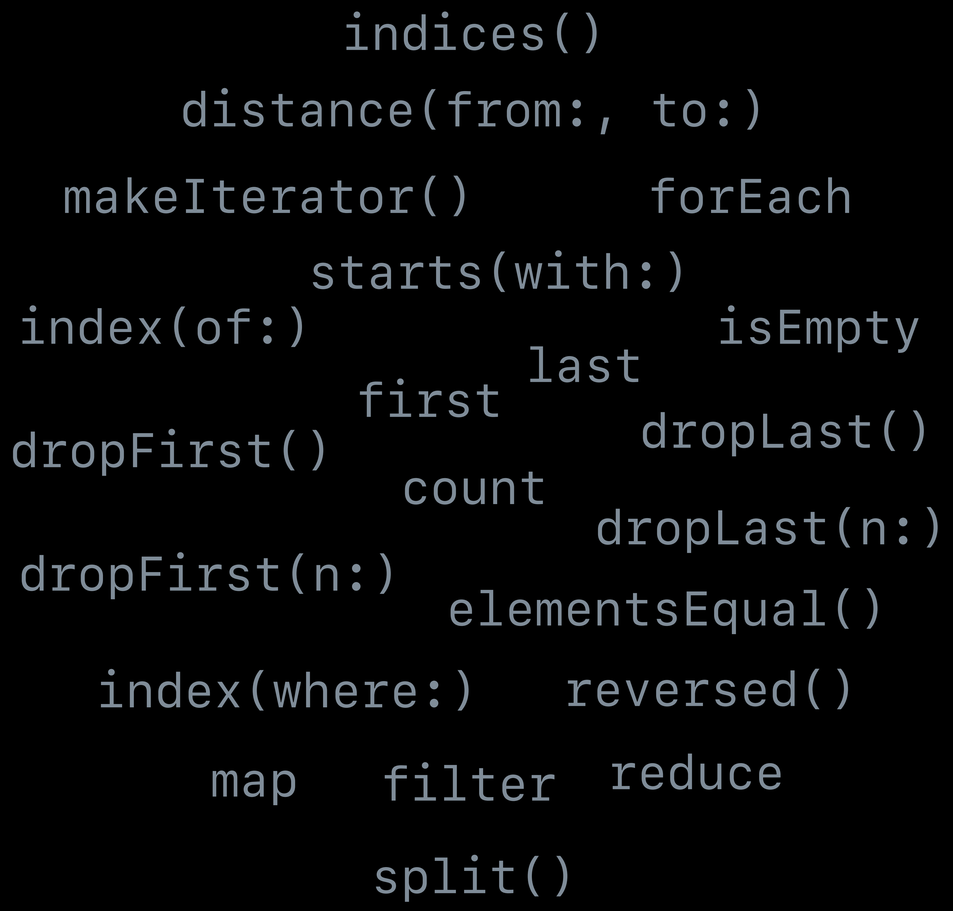
```
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

```swift
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

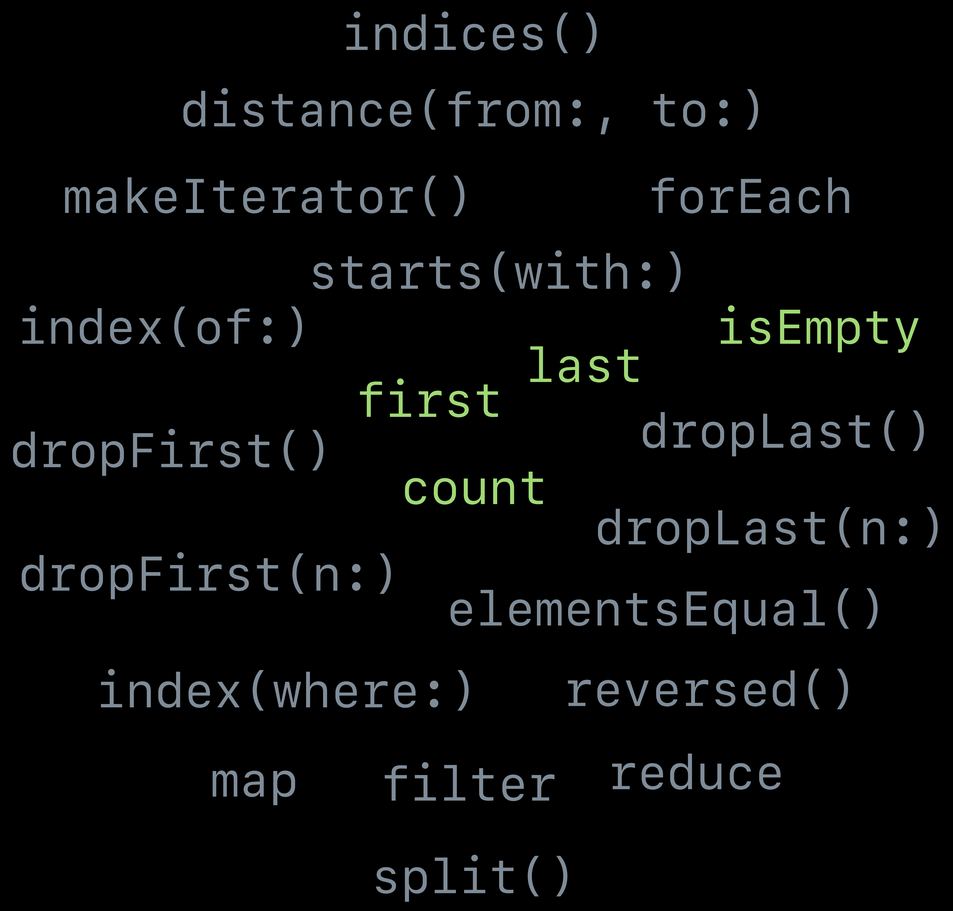
```
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

```
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

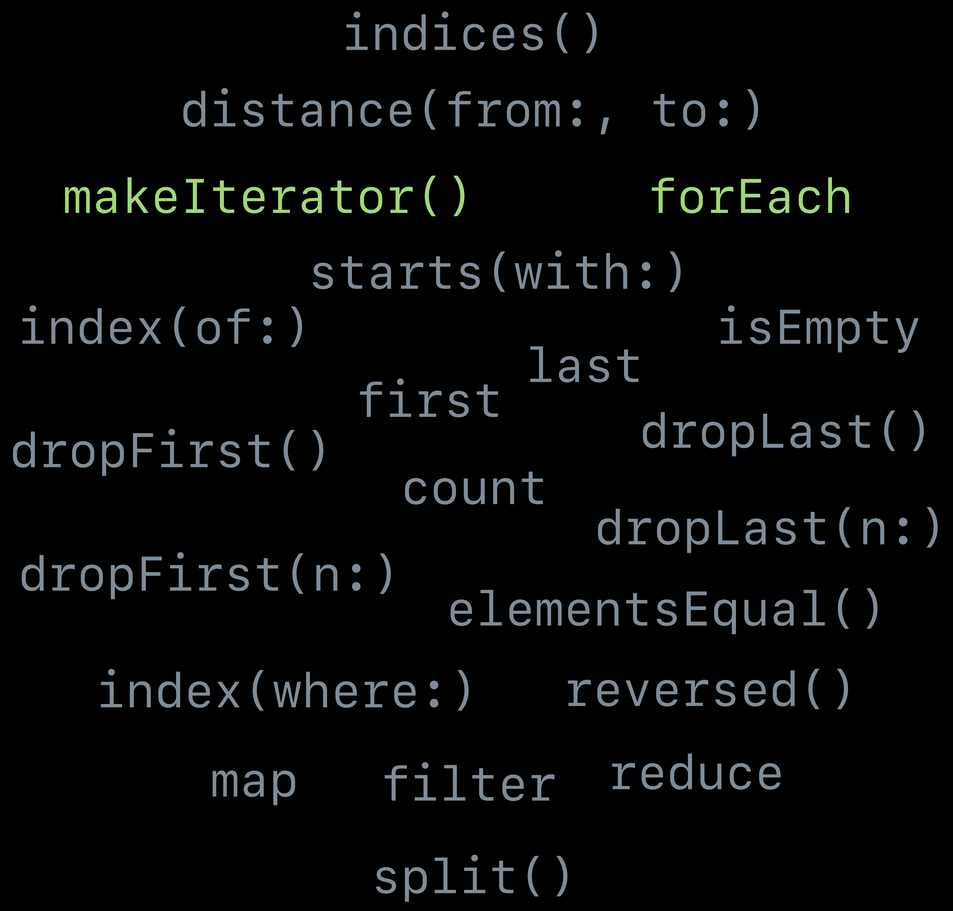
Protocol Extensions



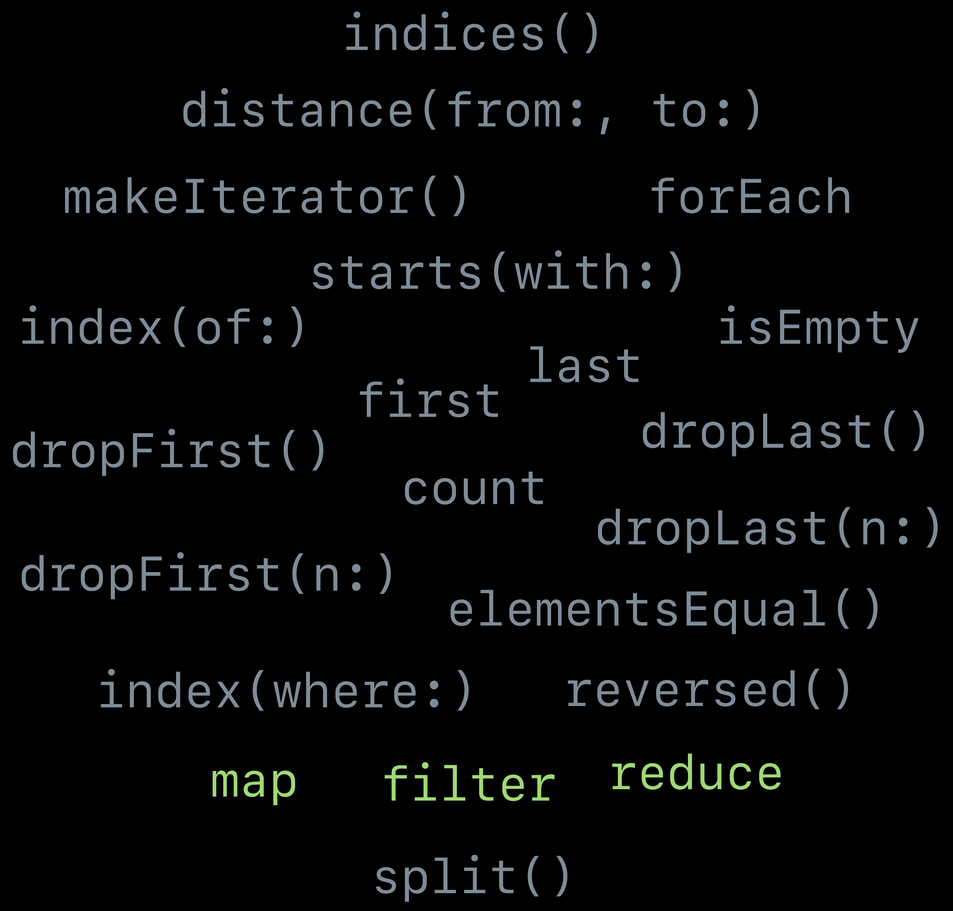
Protocol Extensions



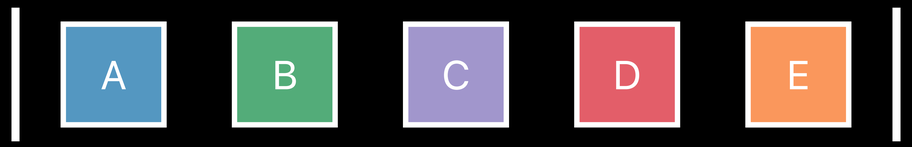
Protocol Extensions



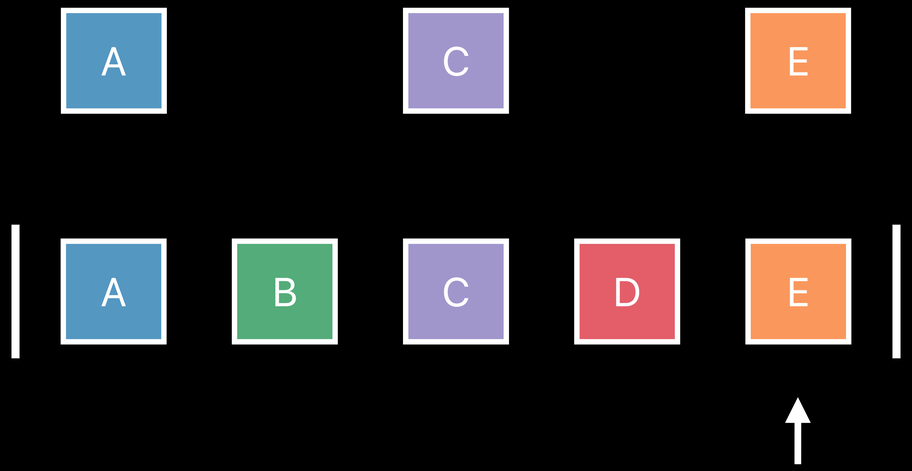
Protocol Extensions



Every Other Element



Every Other Element



```
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) }
```

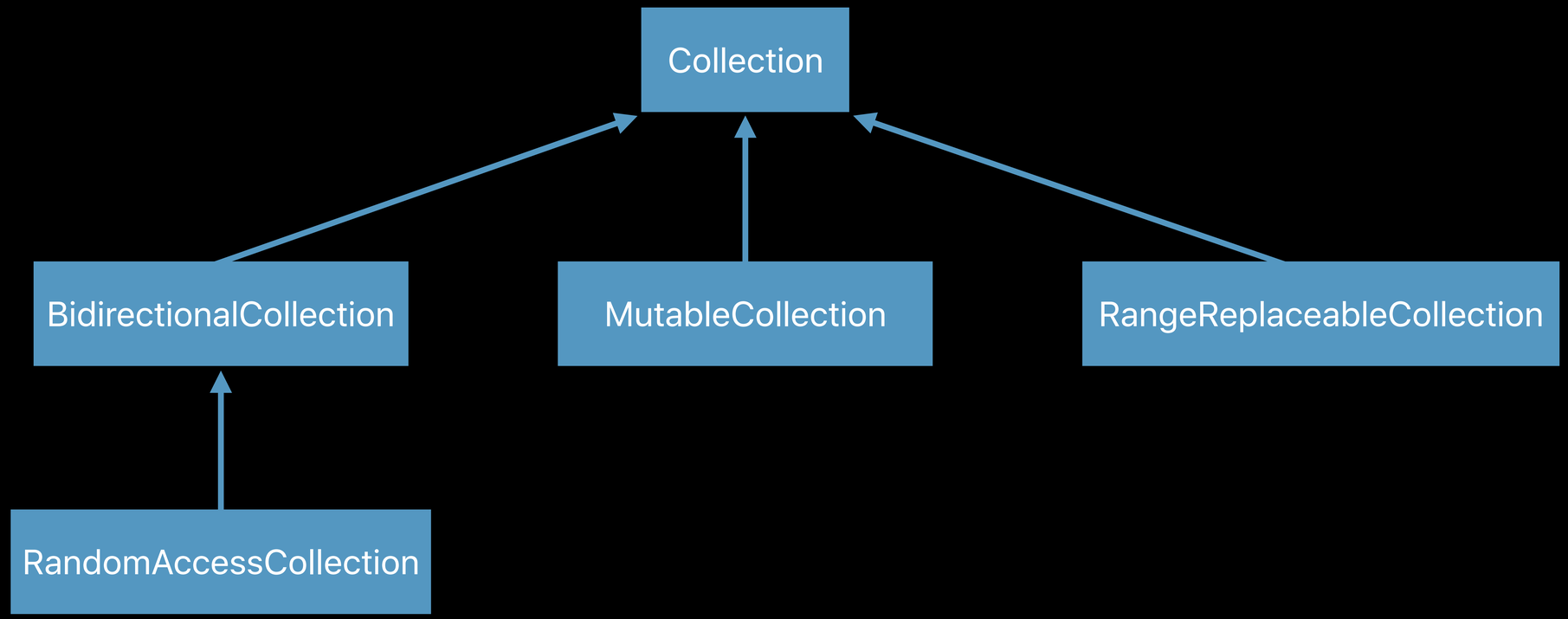
```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) }
```

```swift
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) } // 1, 3, 5, 7, 9
```

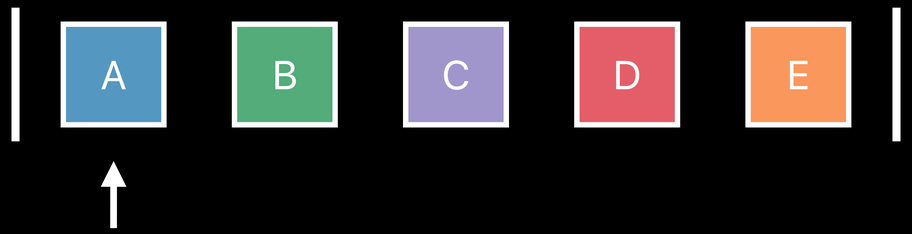
Collections Protocol Hierarchy

Collections Protocol Hierarchy



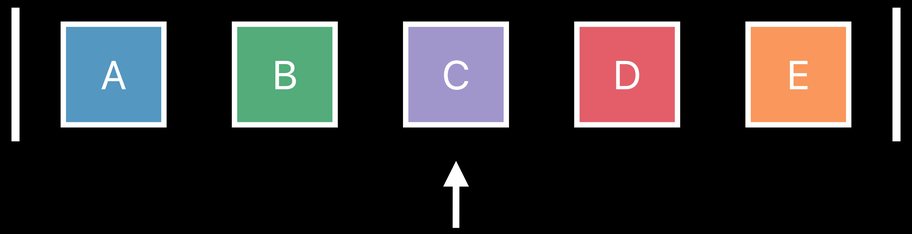
Collections

```
func index(after: Self.Index) -> Self.Index
```



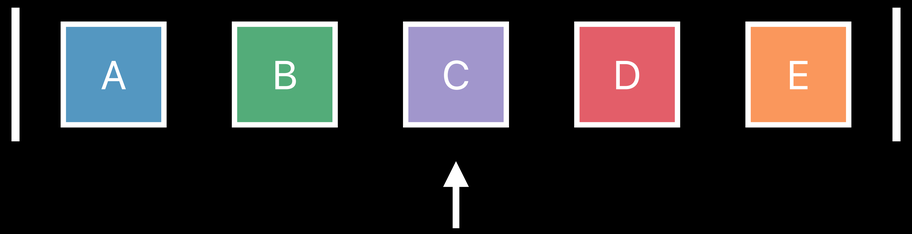
Collections

```
func index(after: Self.Index) -> Self.Index
```



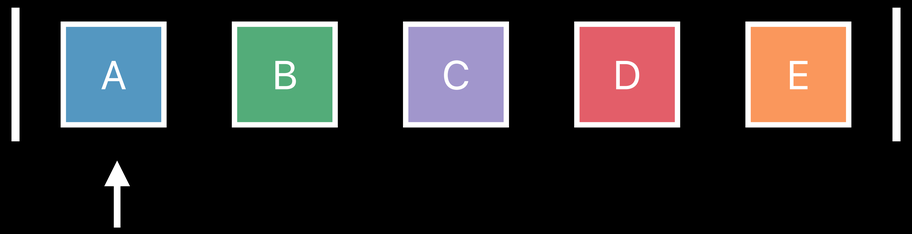
Bidirectional Collections

```
func index(before: Self.Index) -> Self.Index
```



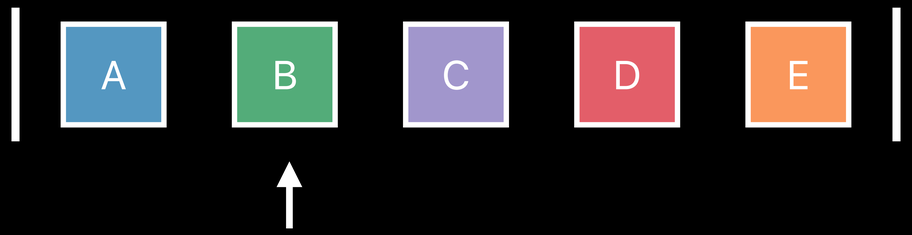
Bidirectional Collections

```
func index(before: Self.Index) -> Self.Index
```



Bidirectional Collections

```
func index(before: Self.Index) -> Self.Index
```

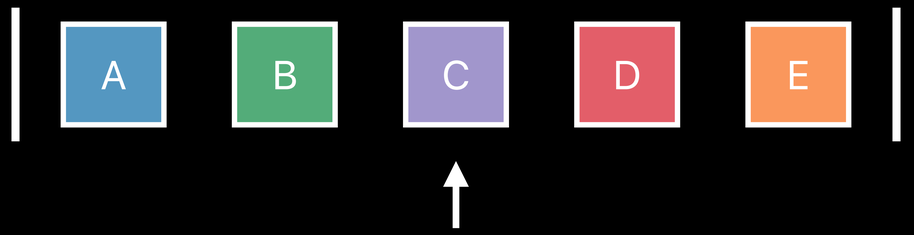


Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```

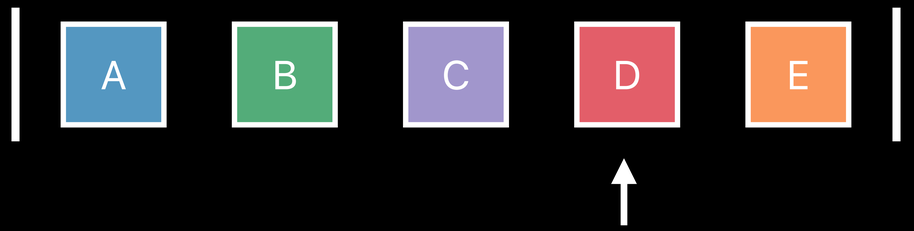
Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```



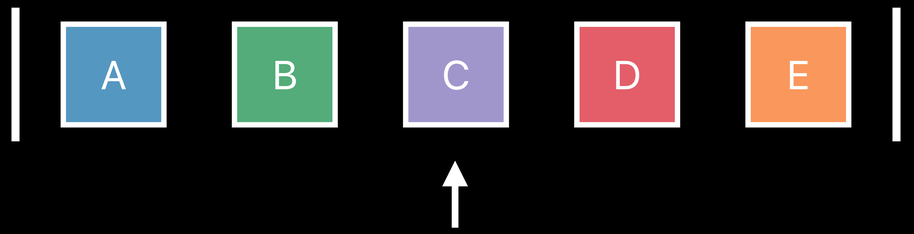
Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```



Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```



Collections In Swift

`Array` `Set` `Dictionary`

Collections In Swift

Array Set Dictionary

Data Range String

Collections In Swift

`Array` `Set` `Dictionary`

`IndexPath` `IndexSet`

`Data` `Range` `String`

Collections In Swift

NSSet

NSArray

NSDictionary

Set

Array

Dictionary

NSPointerArray

IndexPath

IndexSet

NSData

NSHashTable

String

Data

ContiguousArray

Range

NSIndexSet

NSCountedSet

Indices

Each collection defines its own index

Must be `Comparable`

Think of these as opaque

What is the first element of an Array?

```
array[0]
```

What is the first element of a Set?

```
set[0]
```

```
set[0]
```

Cannot subscript a value of type 'Set' with an index of type 'Int'

```
set[set.startIndex]
```

```
set[set.startIndex]
array[array.startIndex]
```

```
set[set.startIndex]
array[array.startIndex]
```

Fatal error: Index out of range

```
array.first
set.first
```

What is the second element of a Collection?

Find the Second Element of a `Collection`

```
extension Collection {
    var second: Element? {

    }
}
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        return self[1]
    }
}
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        return self[1]
    }
}
```

```
Cannot subscript a value of type 'Collection'
          with an index of type 'Int'
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        return self[self.startIndex + 1]
    }
}
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        return self[self.startIndex + 1]
    }
}
```

Binary operator '+' cannot be applied to operands of type 'Index' and 'Int'

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?

        // Get the second index

        // Is that index valid?

        // Return the second element

    }
}
```

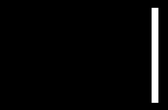
Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?

        // Get the second index

        // Is that index valid?

        // Return the second element

    }
}
```


startIndex
endIndex

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index

        // Is that index valid?

        // Return the second element

    }
}
```



startIndex
endIndex

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index

        // Is that index valid?

        // Return the second element

    }
}
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?

        // Return the second element

    }
}
```

Find the Second Element of a Collection

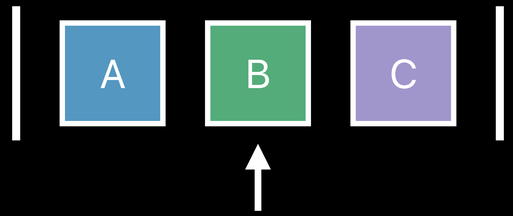
```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?

        // Return the second element

    }
}
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element

    }
}
```

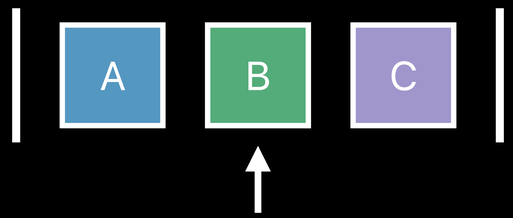
Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element

    }
}
```



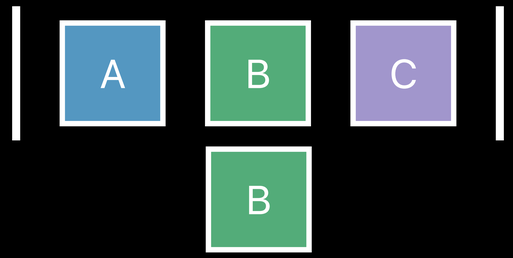
Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element
        return self[index]
    }
}
```



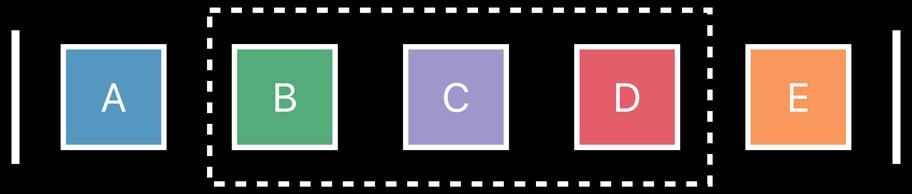
Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element
        return self[index]
    }
}
```

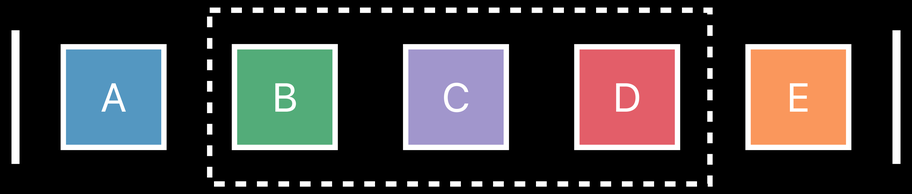


Forming a Slice

Forming a Slice

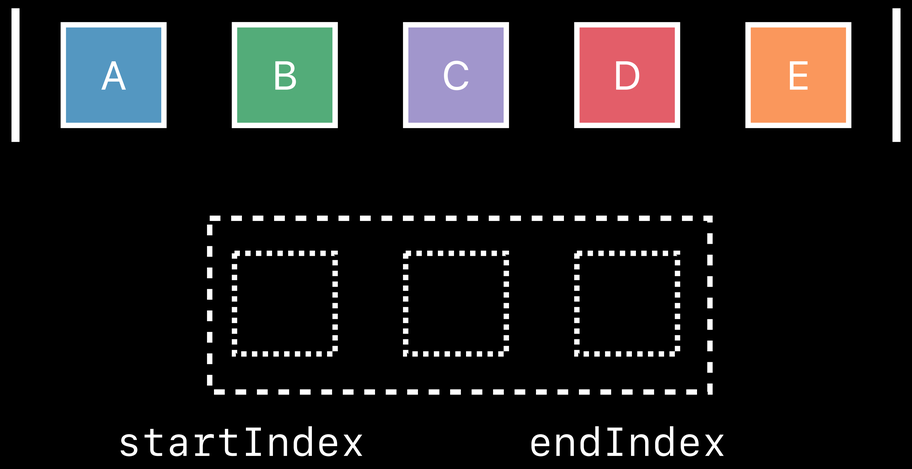


Forming a Slice

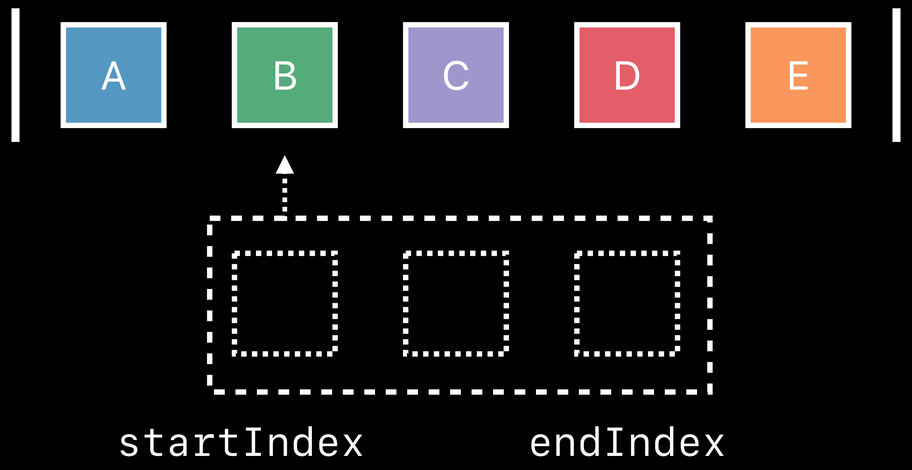


`startIndex` `endIndex`

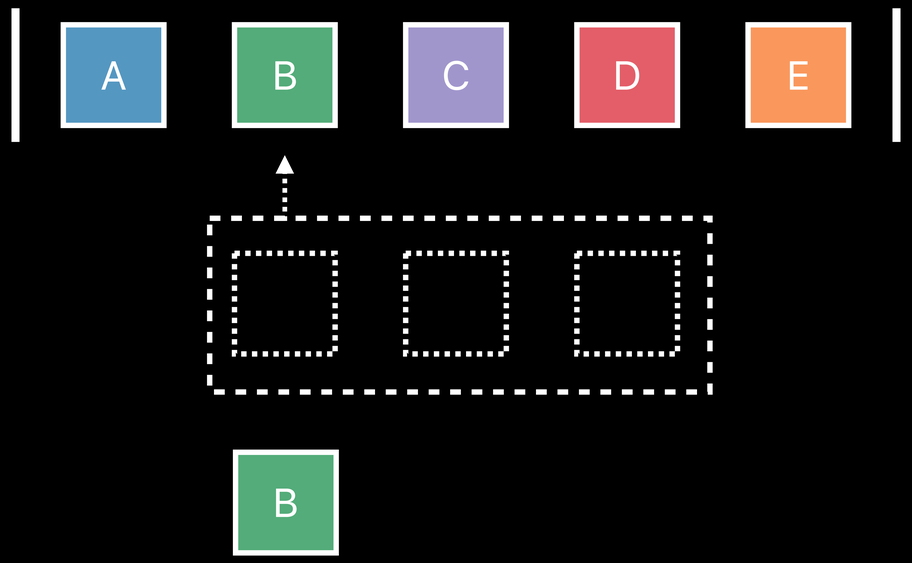
Forming a Slice



Forming a Slice



Forming a Slice




```
subscript[index]
```

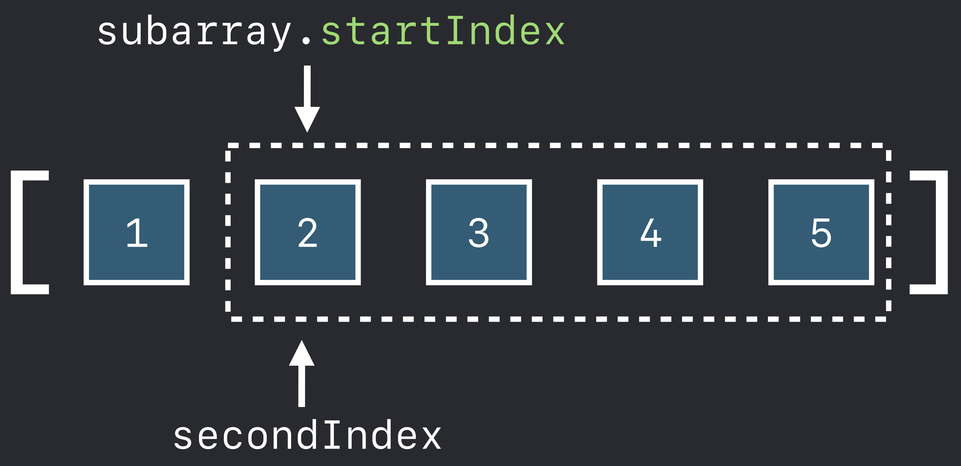
```
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex) // true
```



Find the Second Element of a Collection

```
var second: Element? {
    // Is the collection empty?
    guard self.startIndex != self.endIndex else { return nil }
    // Get the second index
    let index = self.index(after: self.startIndex)
    // Is the second index valid?
    guard index != self.endIndex else { return nil }
    // Return the second element
    return self[index]
}
```

Find the Second Element of a Collection

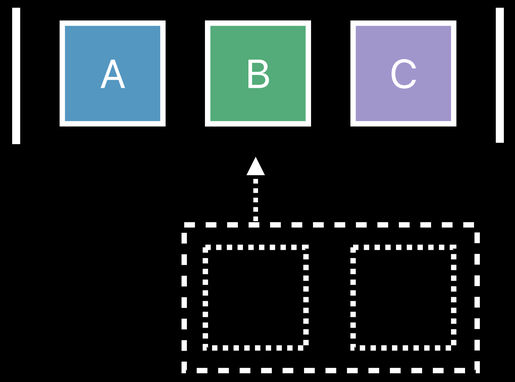
```
var second: Element? {
   return self.dropFirst().first
}
```

Find the Second Element of a Collection

```
var second: Element? {
   return self.dropFirst().first
}
```

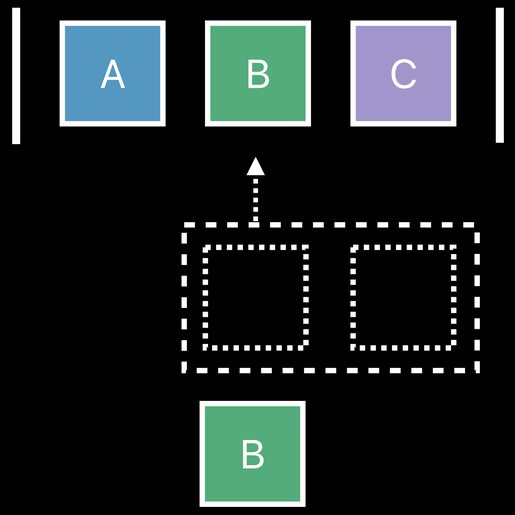
Find the Second Element of a Collection

```
var second: Element? {
    return self.dropFirst().first
}
```



Find the Second Element of a Collection

```
var second: Element? {
   return self.dropFirst().first
}
```



Slices

Produce `Collection`-like peers of original collection

Slices

Produce `Collection`-like peers of original collection

Slices

Produce `Collection`-like peers of original collection

`Array` ⇢ `ArraySlice`

`String` ⇢ `Substring`

Slices

Produce `Collection`-like peers of original collection

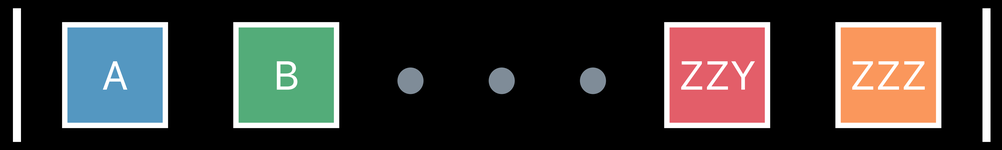
`Array`	--▸	`ArraySlice`
`String`	--▸	`Substring`
`Set`	--▸	`Slice<Set>`

Slices

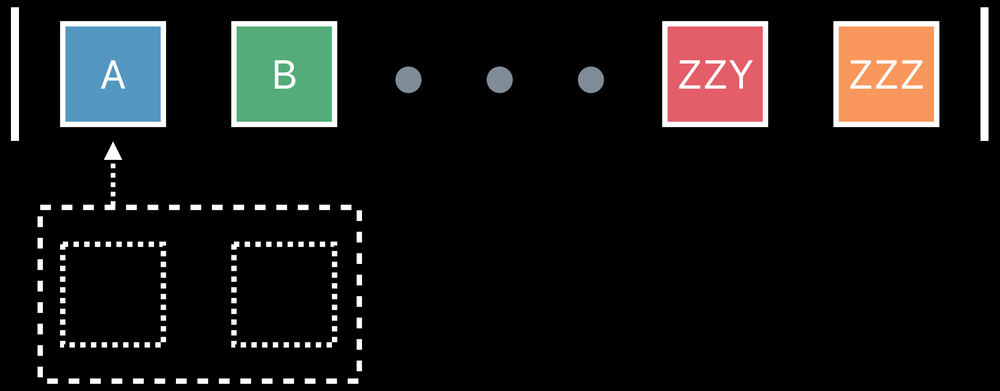
Produce `Collection`-like peers of original collection

`Array`	--▸	`ArraySlice`
`String`	--▸	`Substring`
`Set`	--▸	`Slice<Set>`
`Data`	--▸	`Data`
`Range`	--▸	`Range`

Slices Keep Underlying Storage Alive



Slices Keep Underlying Storage Alive



```
// Slicing Keeps Underlying Storage

extension Array {
    var firstHalf: ArraySlice<Element> {
        return self.dropLast(self.count / 2)
    }
}
```

```
// Slicing Keeps Underlying Storage

extension Array {
    var firstHalf: ArraySlice<Element> {
        return self.dropLast(self.count / 2)
    }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf
array = []
```

```
// Slicing Keeps Underlying Storage

extension Array {
    var firstHalf: ArraySlice<Element> {
        return self.dropLast(self.count / 2)
    }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []
```

```
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []
```

```
// Slicing Keeps Underlying Storage

extension Array {
    var firstHalf: ArraySlice<Element> {
        return self.dropLast(self.count / 2)
    }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1
```

```swift
// Slicing Keeps Underlying Storage

extension Array {
    var firstHalf: ArraySlice<Element> {
        return self.dropLast(self.count / 2)
    }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1

let copy = Array(firstHalf) // [1, 2, 3, 4]
firstHalf = []
print(copy.first!)
```

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1

let copy = Array(firstHalf) // [1, 2, 3, 4]
firstHalf = []
print(copy.first!)
```

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1

let copy = Array(firstHalf) // [1, 2, 3, 4]
firstHalf = []
print(copy.first!) // 1
```

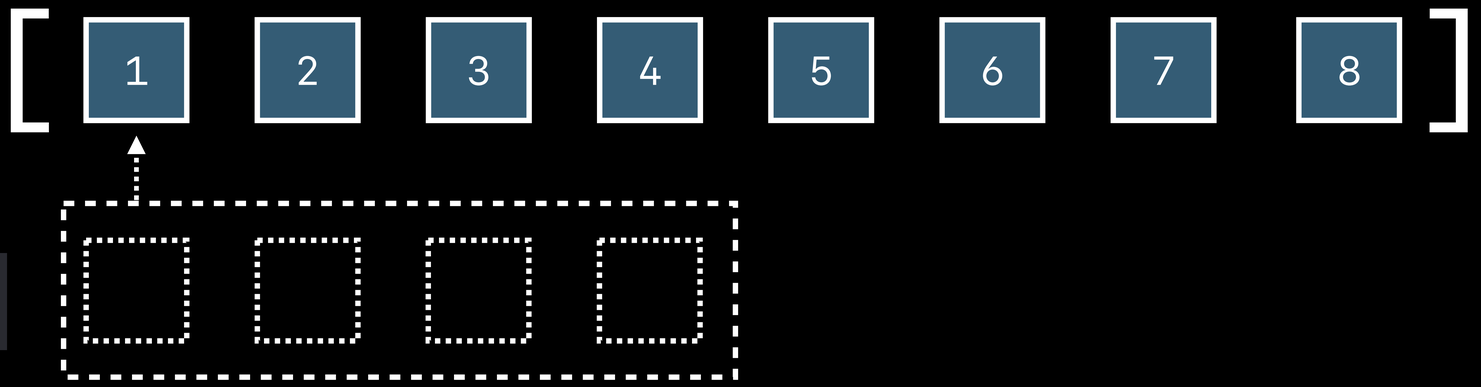
Copying a Slice

```
var array = [...]
```

Copying a Slice

```
var array = [...]
```

```
var firstHalf = array.firstHalf
```

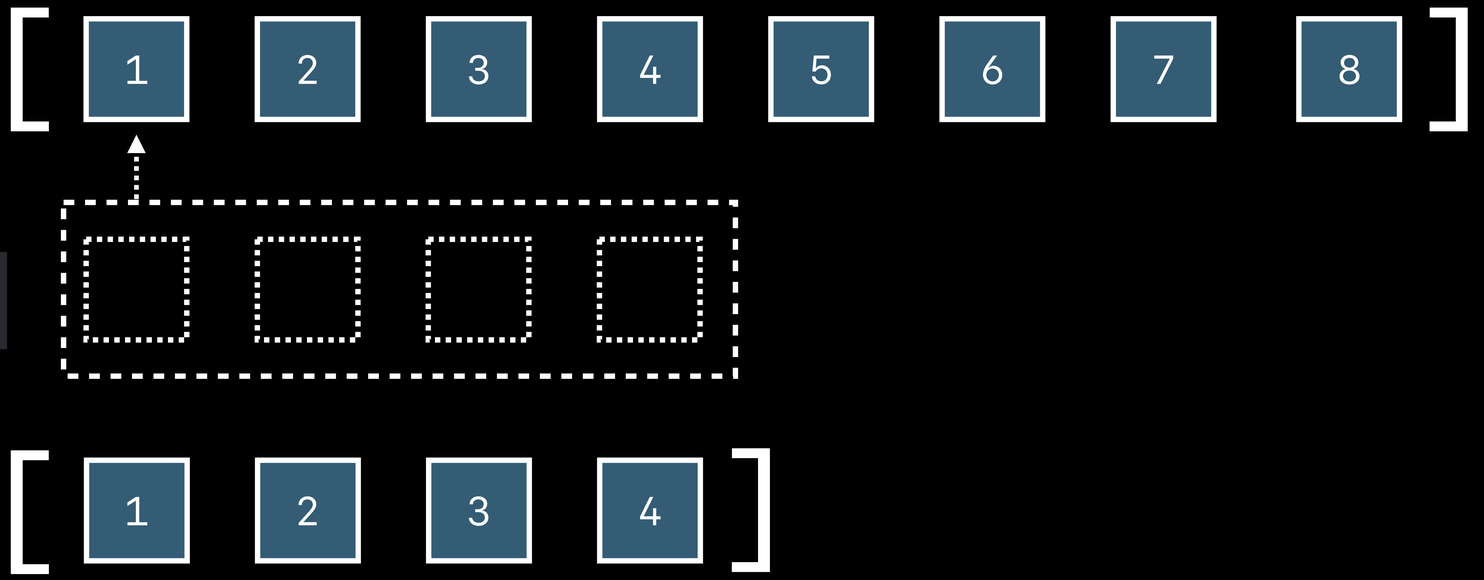


Copying a Slice

```
var array = [...]
```

```
var firstHalf = array.firstHalf
```

```
let copy = Array(firstHalf)
```

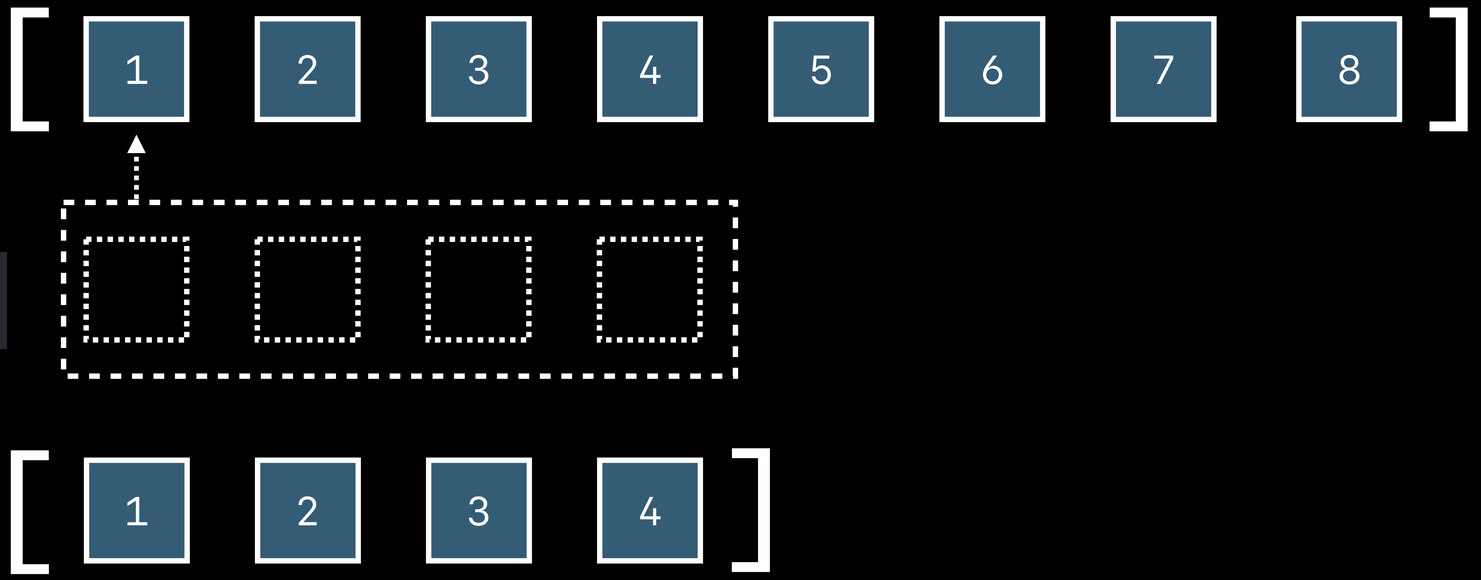


Copying a Slice

```
var array = []
```

```
var firstHalf = array.firstHalf
```

```
let copy = Array(firstHalf)
```

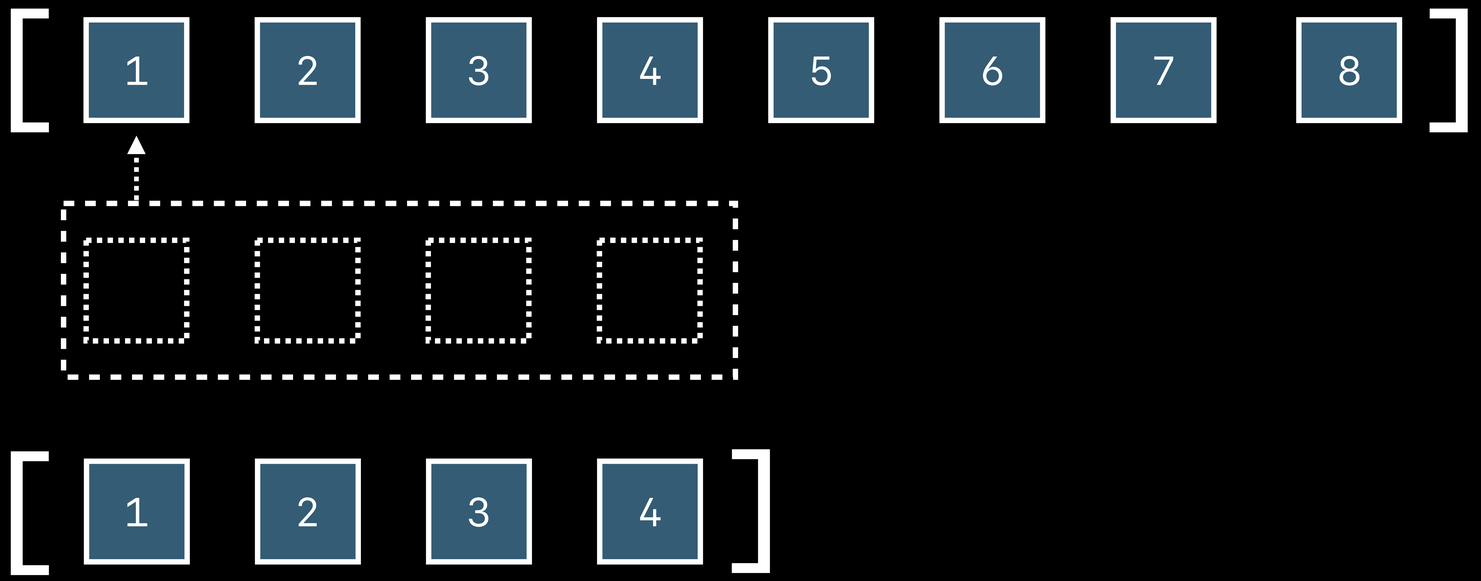


Copying a Slice

```
var array = []
```

```
var firstHalf = []
```

```
let copy = Array(firstHalf)
```



Copying a Slice

```
var array = []
```

```
var firstHalf = []
```

```
let copy = Array(firstHalf)
```

Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```

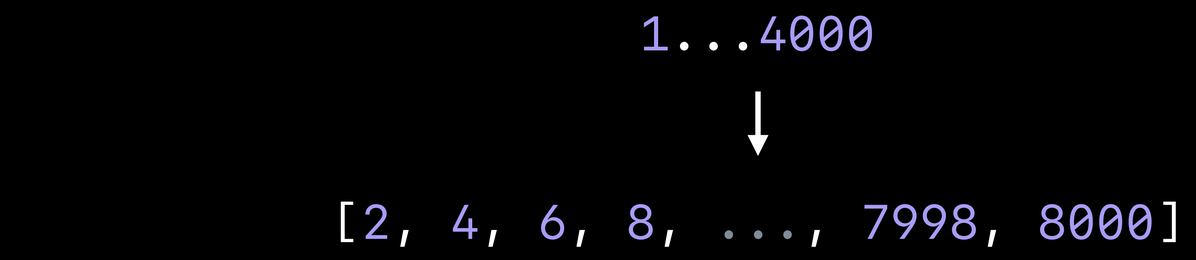
Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```

1...4000

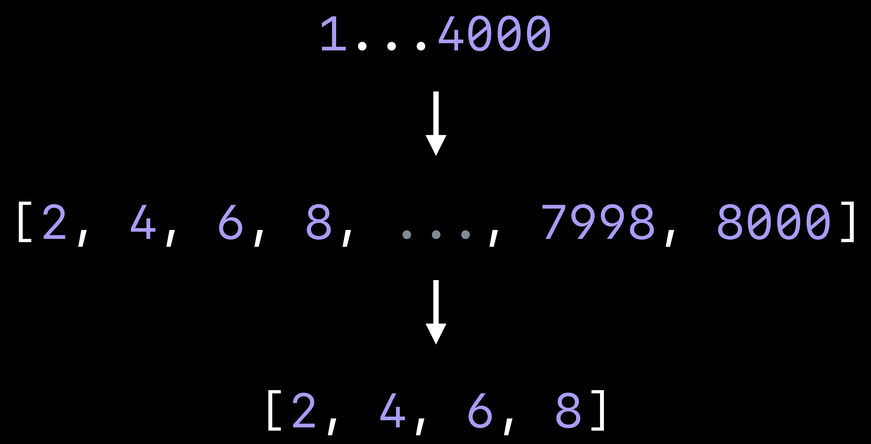
Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```



Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```



Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```

```
[2, 4, 6, 8]
```

Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

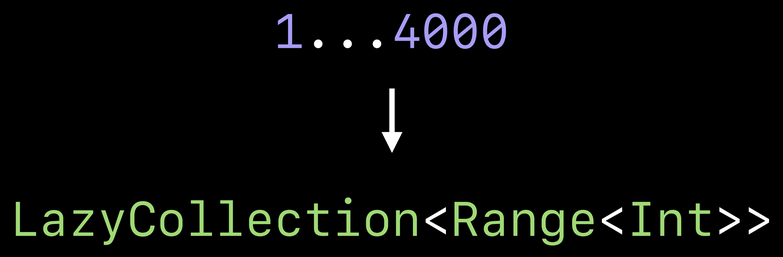
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

1...4000

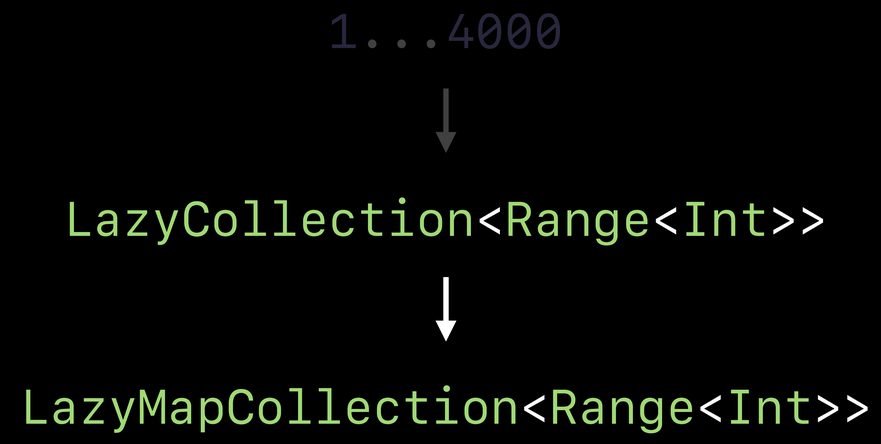
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```



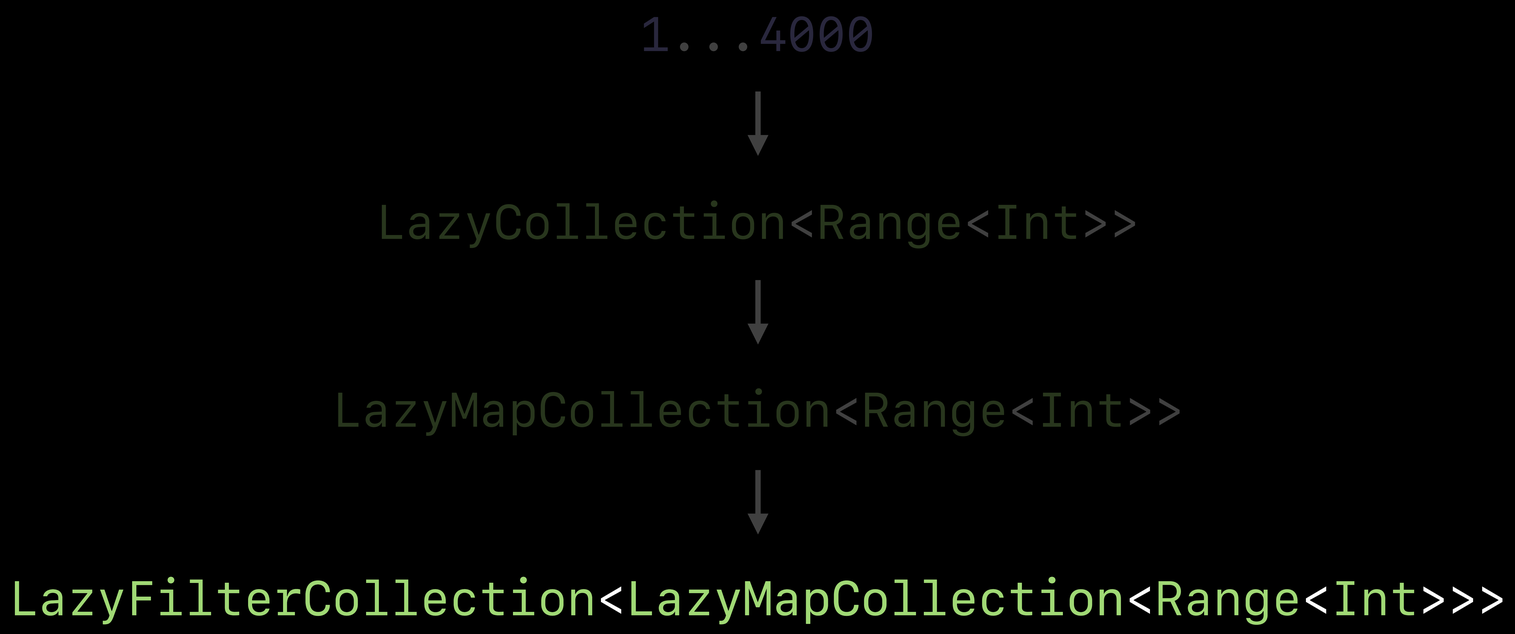
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

LazyFilterCollection LazyMapCollection 1...4000

Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```

Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```

.first

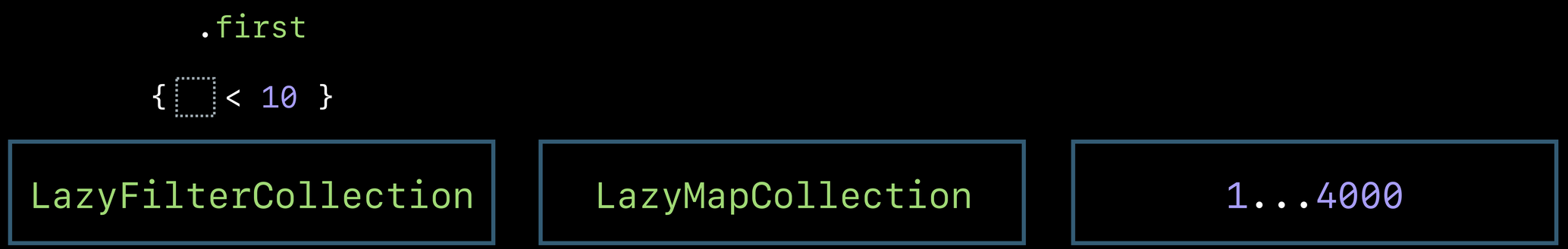
LazyFilterCollection

LazyMapCollection

1...4000

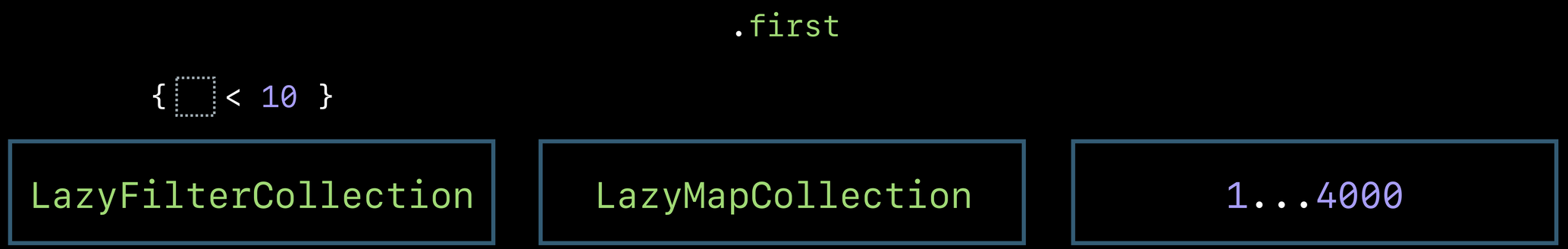
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```

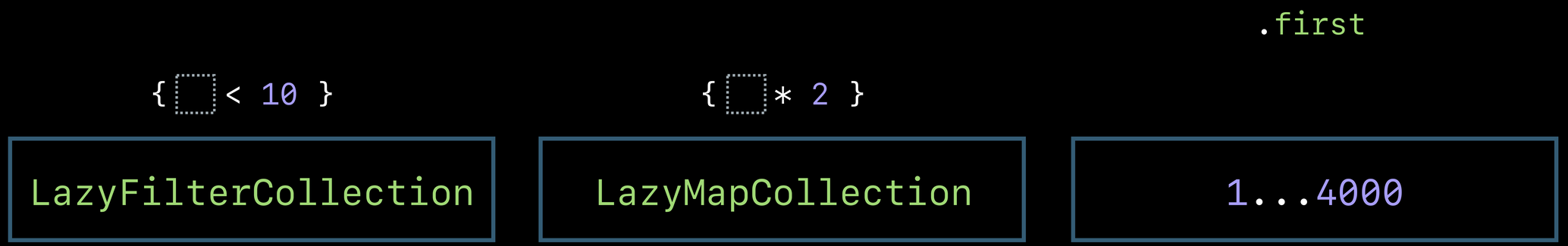


Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```

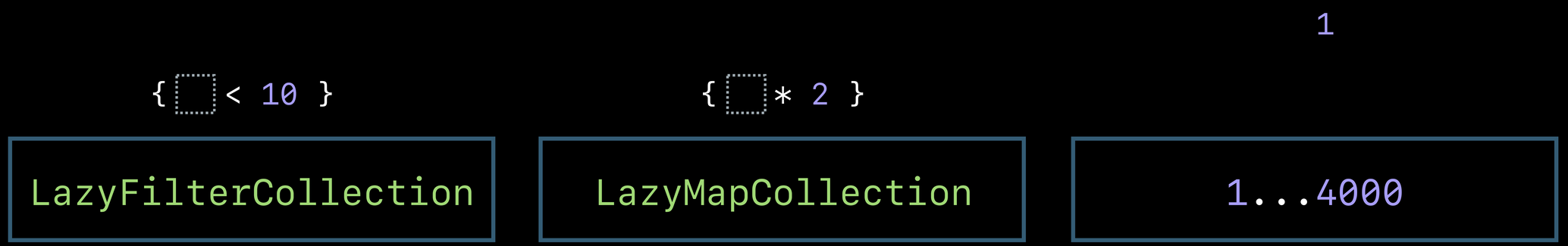
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



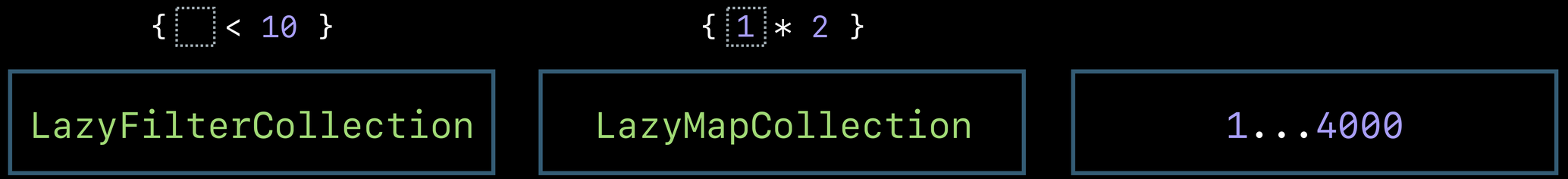
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



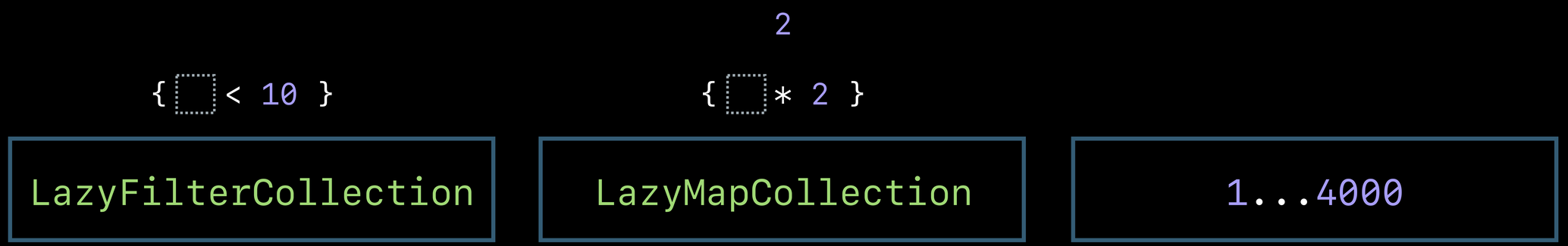
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



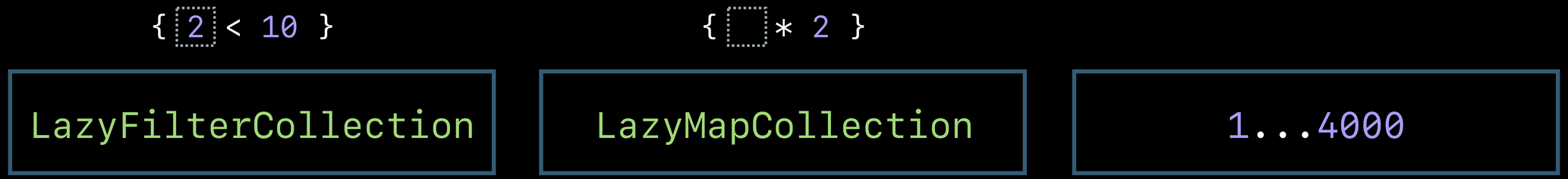
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



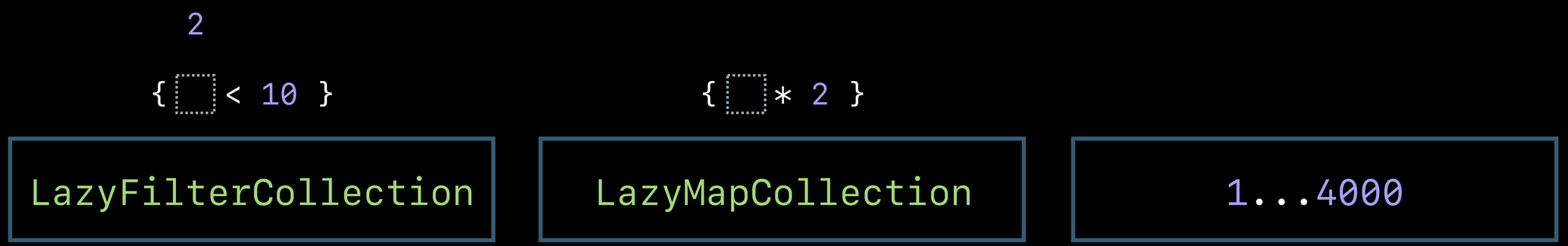
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



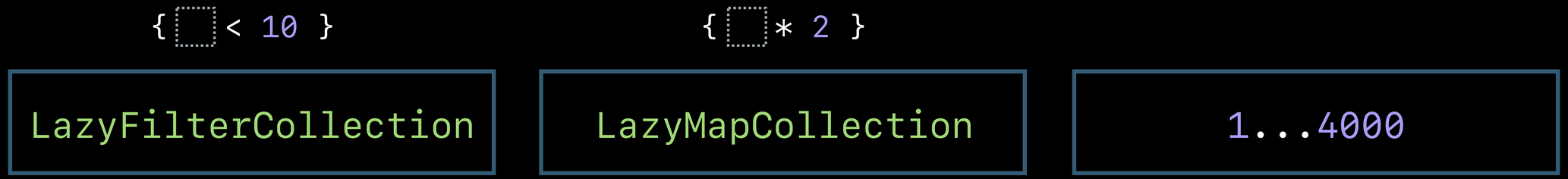
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first    // 2
```



```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter { // LazyFilterCollection<Array<String>>
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
                  ↓
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Grizzly'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Grizzly'
    return $0.contains("Bear") // false
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Panda'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Panda'
    return $0.contains("Bear") // false
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Spectacled'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Spectacled'
    return $0.contains("Bear") // false
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Gummy Bears'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Gummy Bears'
    return $0.contains("Bear") // true
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!)
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!)
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!) // Gummy Bears
```

```
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears)
print(filteredBears.first!)
```

```
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears)
print(filteredBears.first!)
```

```
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears) // ["Gummy Bears"]
print(filteredBears.first!)
```

```
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears) // ["Gummy Bears"]
print(filteredBears.first!) // Gummy Bears
```

Advice: When to Be Lazy?

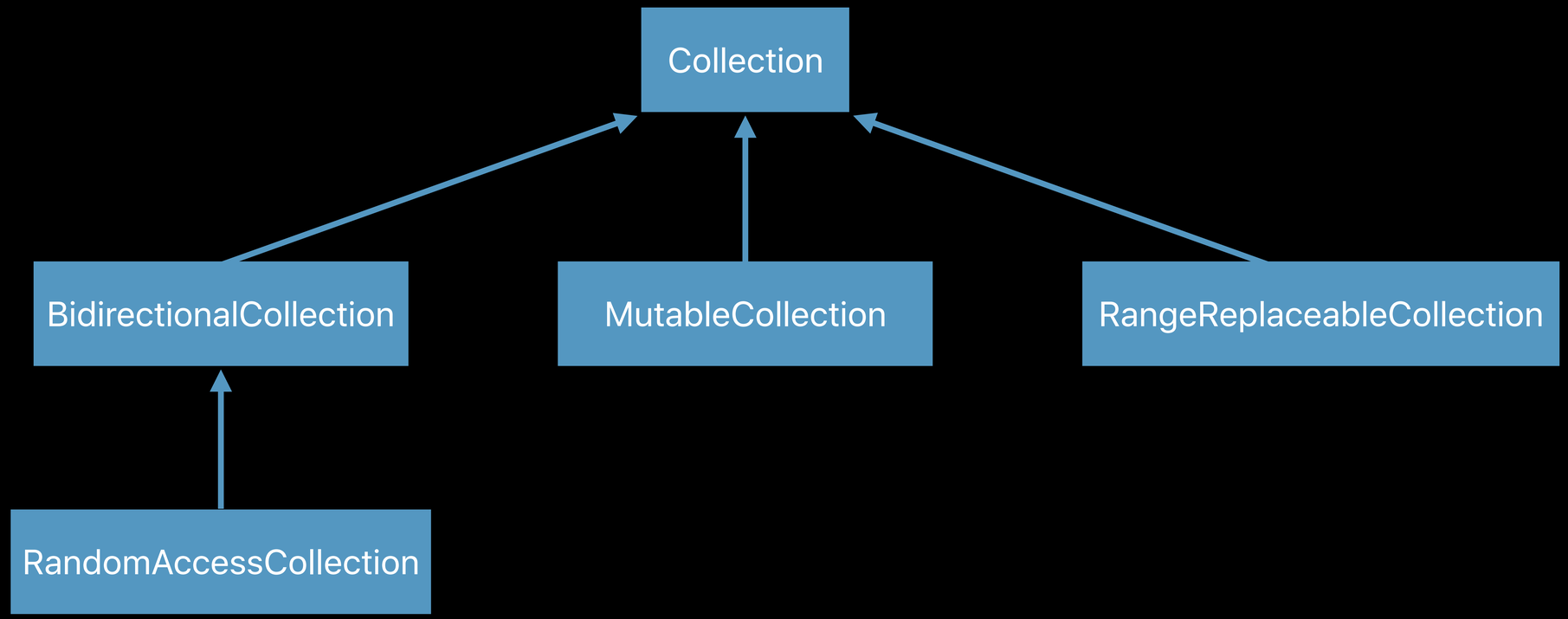
Chained computation

Only need part of a result

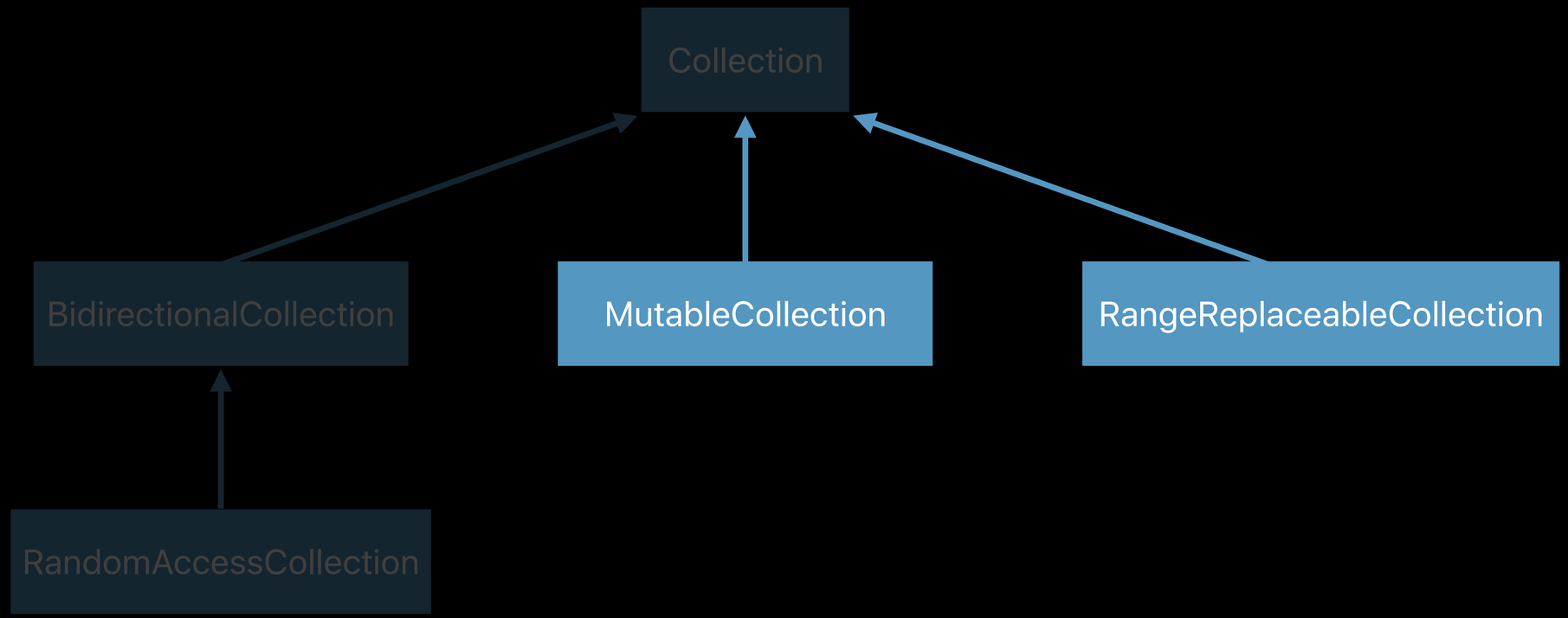
No side effects

Avoid API boundaries

Collections Protocol Hierarchy

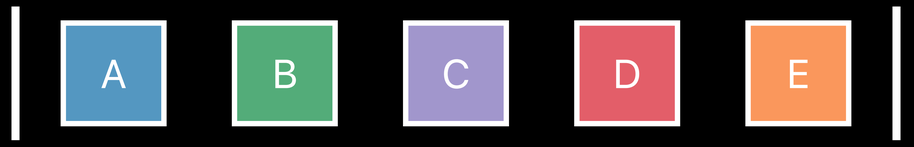


Collections Protocol Hierarchy



Mutable Collection

```
// constant time
subscript(_: Self.Index) -> Element { get set }
```



Mutable Collection

```
// constant time
subscript(_: Self.Index) -> Element { get set }
```

Range Replaceable Collections

```
replaceSubrange(_:, with:)
```

Range Replaceable Collections

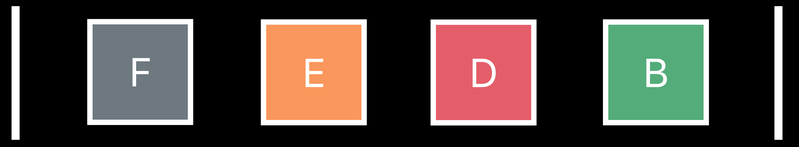
```
replaceSubrange(_:, with:)
```

Range Replaceable Collections

```
replaceSubrange(_:, with:)
```

Range Replaceable Collections

```
replaceSubrange(_:, with:)
```



Why did this collection code crash?

Follow-Up Questions

Are you mutating your collection?

Are your collections accessed from multiple threads?

Crashing Collection Code

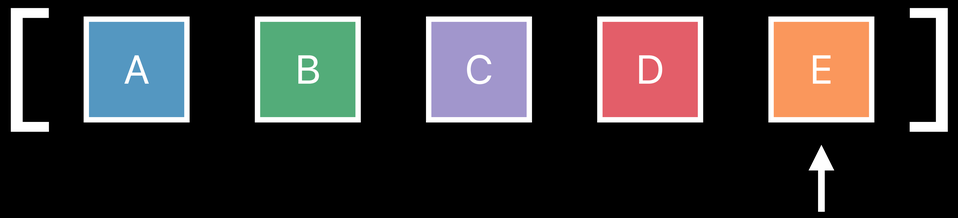
```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

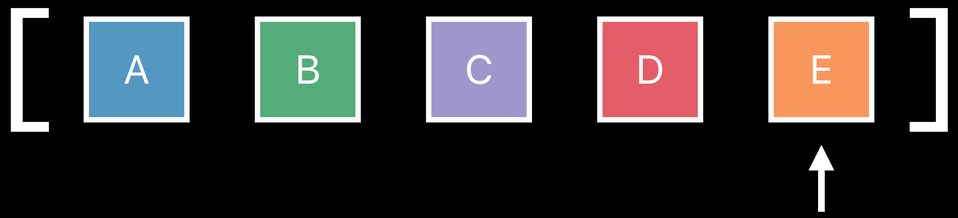
Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```



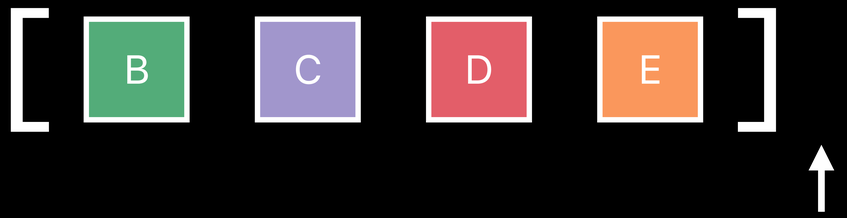
Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```



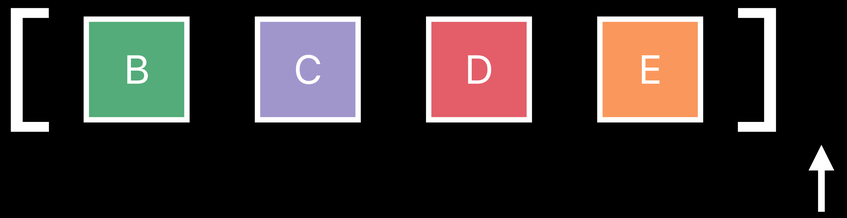
Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```



Crashing Collection Code

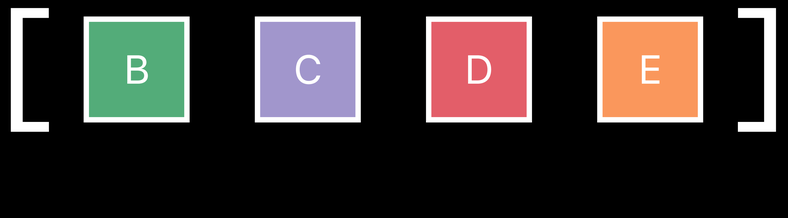
```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```



Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

```
Fatal Error: Index out of range.
```



Avoid Index Invalidation

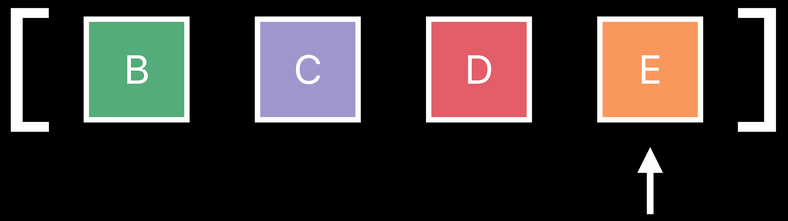
```
var array = ["A", "B", "C", "D", "E"]
array.remove(at: array.startIndex)
if let idx = array.firstIndex(of: "E") {
   print(array[idx])
}
```

Avoid Index Invalidation

```
var array = ["A", "B", "C", "D", "E"]
array.remove(at: array.startIndex)
if let idx = array.firstIndex(of: "E") {
   print(array[idx])
}
```

Avoid Index Invalidation

```
var array = ["A", "B", "C", "D", "E"]
array.remove(at: array.startIndex)
if let idx = array.firstIndex(of: "E") {
   print(array[idx])
}
```



```
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex])

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```swift
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex])

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```swift
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex]) // (key: "sleep", value: "hibernation")
```

```swift
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex]) // (key: "sleep", value: "hibernation")
```

```
Fatal error: Attempting to access Dictionary
elements using an invalid Index
```

```swift
// Always Work With Up-To-Date Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

if let foodIndex = favorites.index(forKey: "food") {
   print(favorites[foodIndex])
}
```

```swift
// Always Work With Up-To-Date Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

if let foodIndex = favorites.index(forKey: "food") {
   print(favorites[foodIndex]) // (key: "food", value: "salmon")
}
```

Advice: Indices and Slices

Use caution when keeping indices/slices

Mutation invalidates

Calculate only as needed

Are your collections reachable from multiple threads?

Multithreaded Mutable Collections

Our collections optimized for single-threaded access

This is a Good Thing™

Undefined behavior without mutual exclusion

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

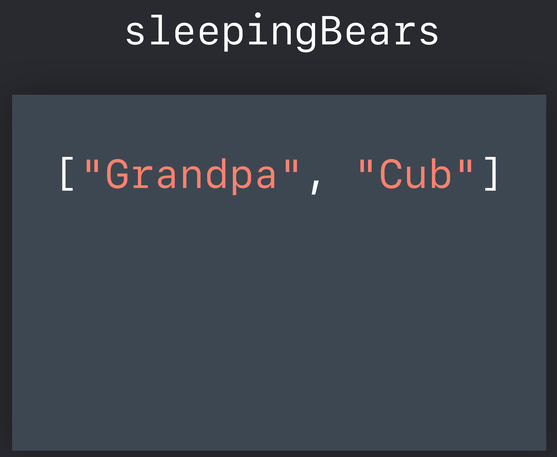
```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```



```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
    ["Grandpa"]
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
    ["Grandpa"]
      ["Cub"]
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
    ["Grandpa"]
      ["Cub"]
```

```
malloc: *** error
for object
0x100586238: pointer
being freed was not
allocated
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

Thread 1

Thread 2

Thread 3

Thread 4

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
    ["Grandpa"]
      ["Cub"]
```

```
malloc: *** error
for object
0x100586238: pointer
being freed was not
allocated
```

```
WARNING: ThreadSanitizer: Swift access race
  Modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 3:
    ...

  Previous modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 2:
    ...

  Location is heap block of size 24 at 0x7b0800023ce0 allocated by main thread:
    ...

SUMMARY: ThreadSanitizer: Swift access race main.swift:515 in closure #1 in gotoSleep()
```

```
WARNING: ThreadSanitizer: Swift access race
  Modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 3:
    ...

  Previous modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 2:
    ...

  Location is heap block of size 24 at 0x7b0800023ce0 allocated by main thread:
    ...

SUMMARY: ThreadSanitizer: Swift access race main.swift:515 in closure #1 in gotoSleep()
```

```
WARNING: ThreadSanitizer: Swift access race
  Modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 3:
     ...

  Previous modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 2:
    ...

  Location is heap block of size 24 at 0x7b0800023ce0 allocated by main thread:
    ...

SUMMARY: ThreadSanitizer: Swift access race main.swift:515 in closure #1 in gotoSleep()
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
queue.async { print(sleepingBears) }
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
queue.async { print(sleepingBears) }
```

```
["Grandpa", "Cub"]
```

Advice: Multithreading

Prefer state accessible from a single thread

When this is not possible:

- Ensure mutual exclusion
- Use TSAN

Advice: Prefer Immutable Collections

Easier to reason about data that can't change

Less surface area for bugs

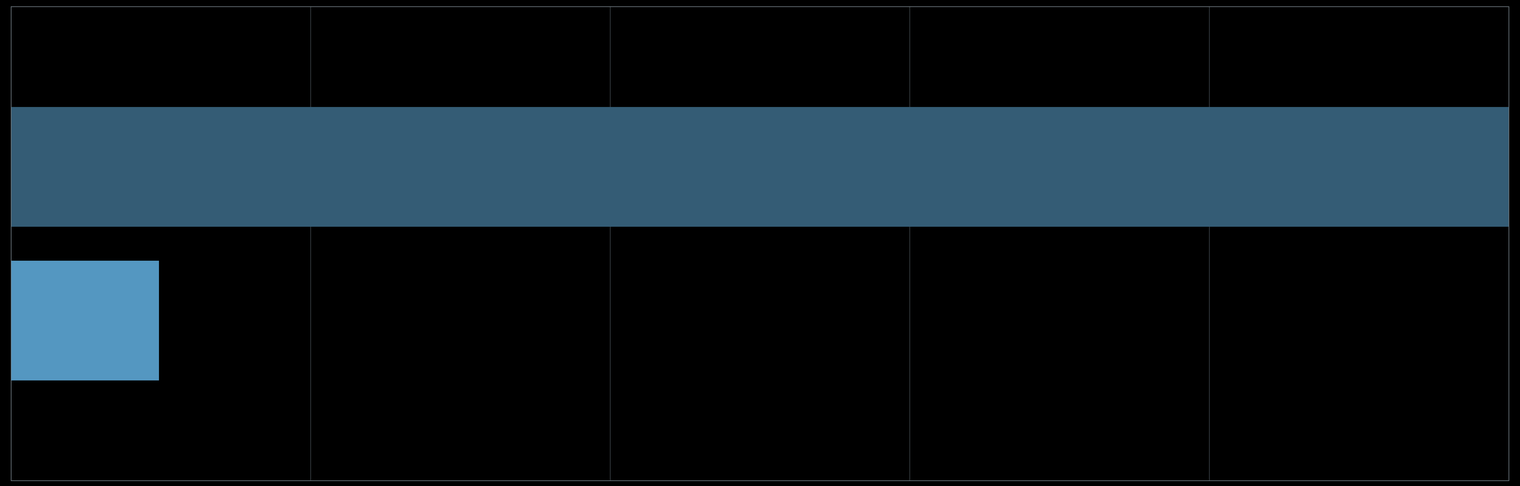
Emulate mutation with slices and lazy

The compiler will help you

Advice: Forming New collections

Use capacity hints if possible

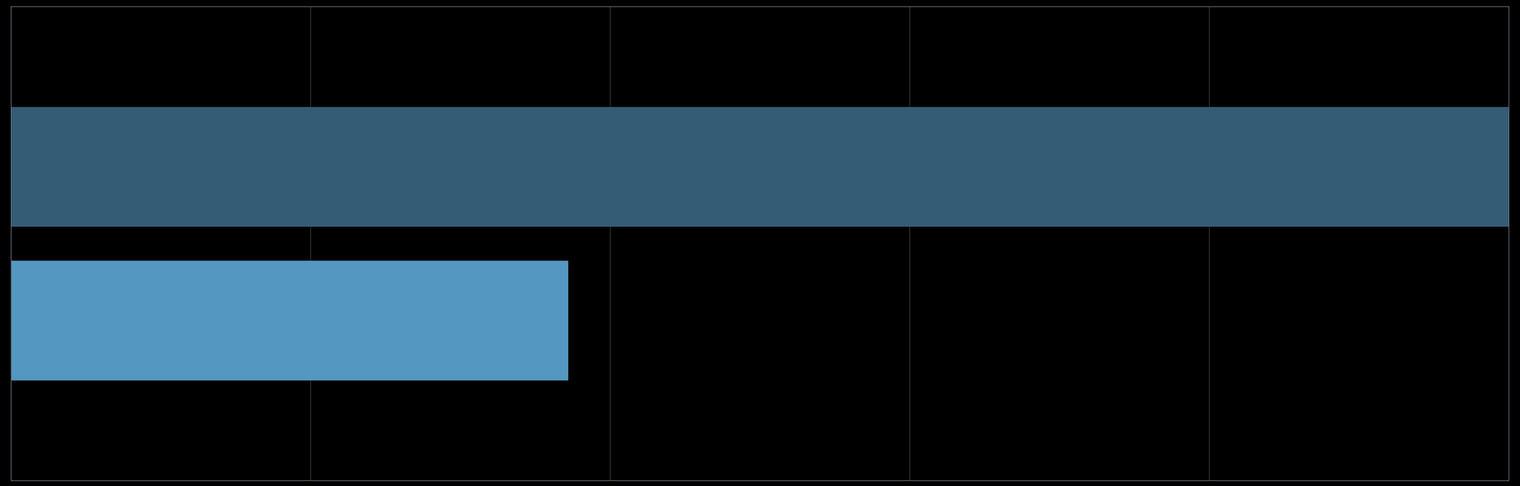
```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Advice: Forming New collections

Use capacity hints if possible

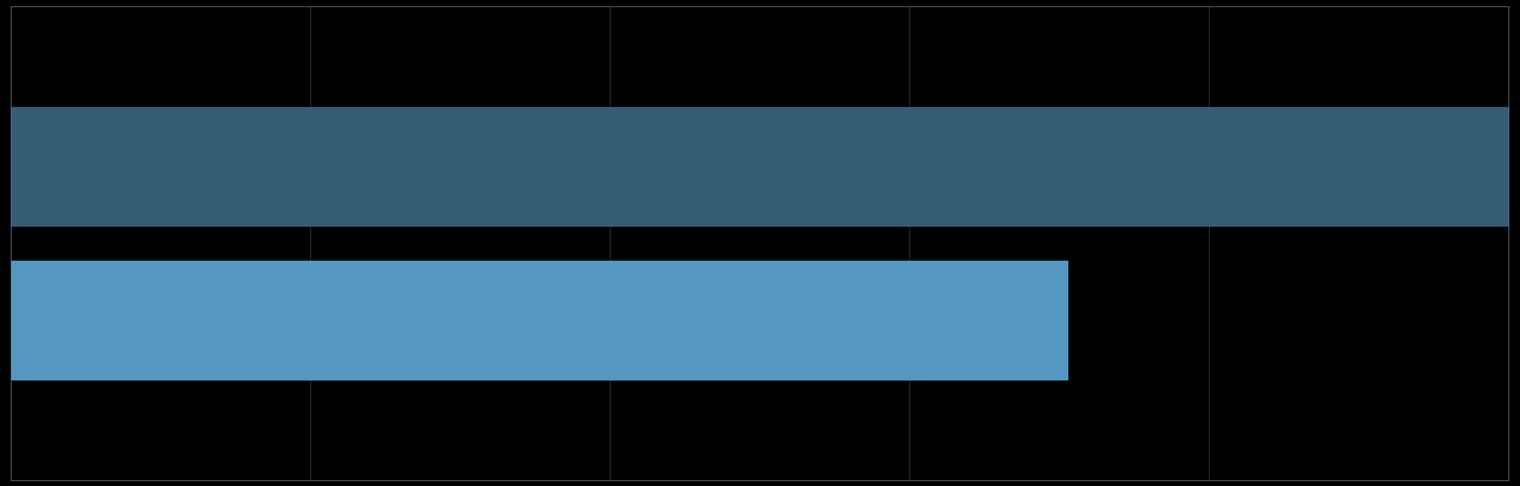
```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Advice: Forming New collections

Use capacity hints if possible

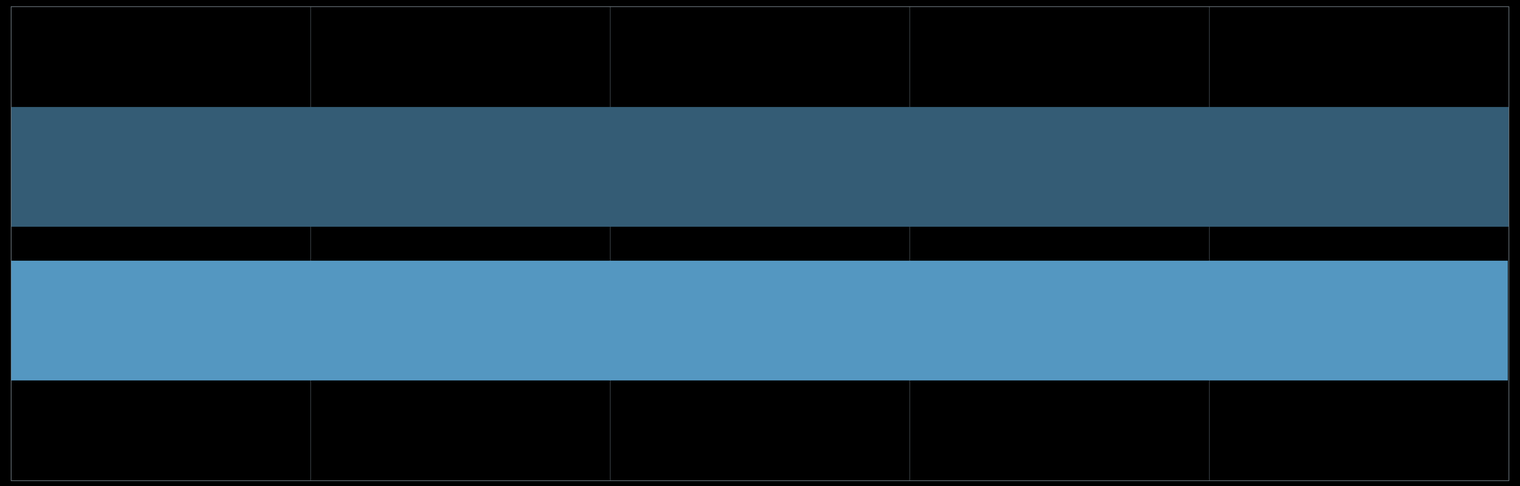
```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Advice: Forming New collections

Use capacity hints if possible

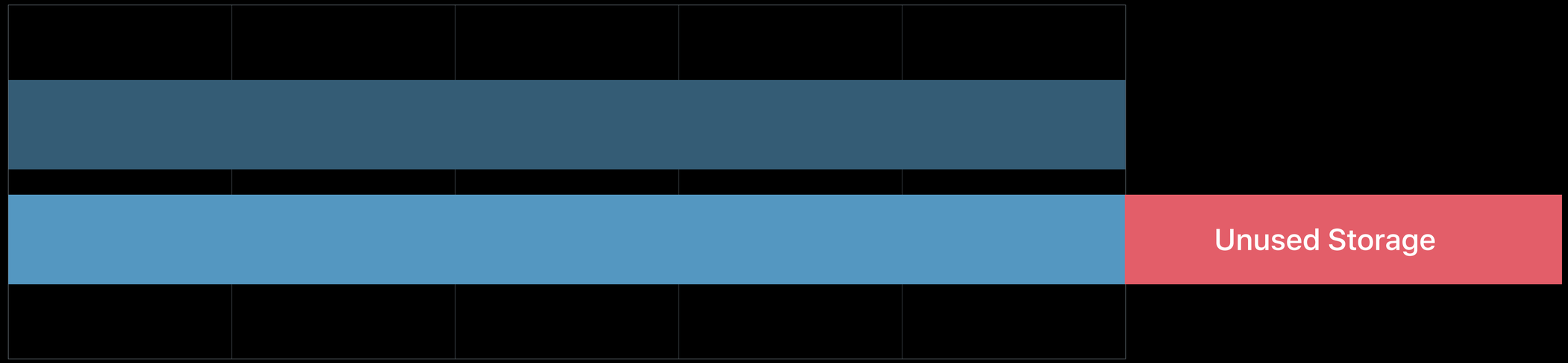
```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Advice: Forming New collections

Use capacity hints if possible

```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Foundation Collections

Reference Type Collections

NSArray
NSMutableArray
NSPointerArray
NSData

NSSet
NSMutableSet
NSCountedSet
NSMutableOrderedSet
NSHashTable
NSIndexSet
NSCharacterSet

NSDictionary
NSMutableDictionary
NSMapTable

Value and Reference Collections

Value and Reference Collections

```
// Value
```

```
// Reference
```

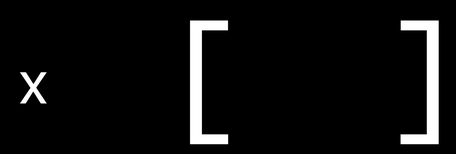
Value and Reference Collections

```swift
// Value
var x: [String] = []
```

```swift
// Reference
let x = NSMutableArray()
```

Value and Reference Collections

```
// Value
var x: [String] = []
```

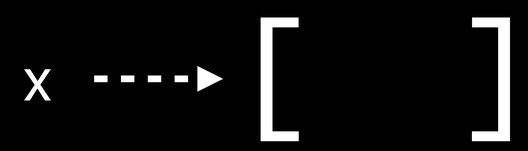



```
// Reference
let x = NSMutableArray()
```

Value and Reference Collections

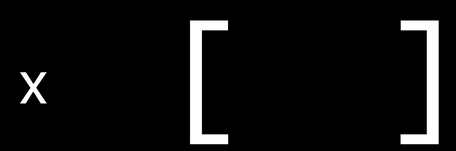
```
// Value
var x: [String] = []
```


```
// Reference
let x = NSMutableArray()
```

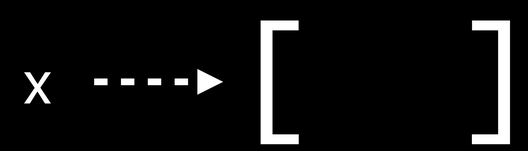


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
```




```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
```



Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
```


```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
```

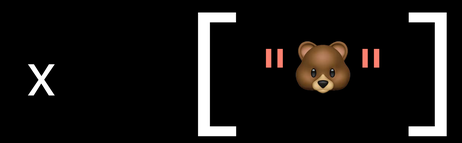
Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
```


```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
```

Value and Reference Collections

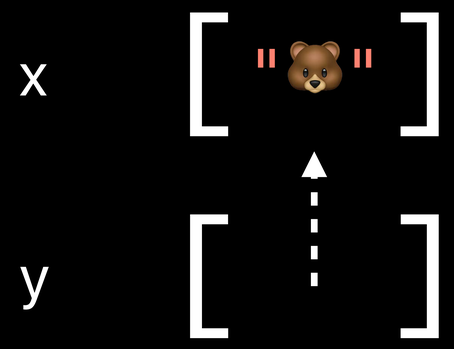
```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
```




```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
```

Value and Reference Collections

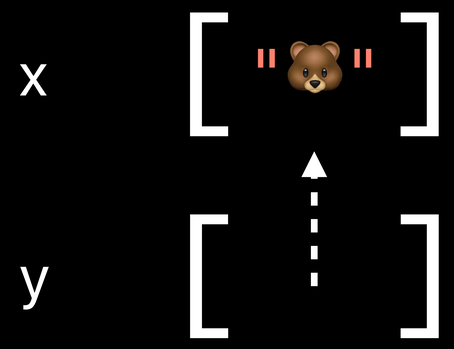
```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
```



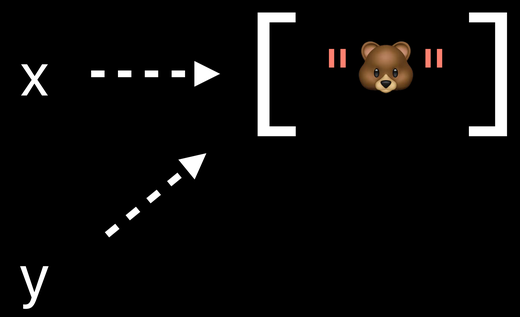

```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
```

Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
```

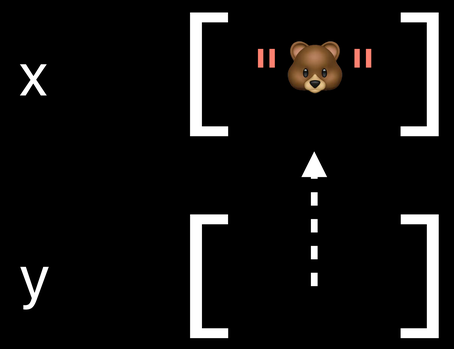



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
```

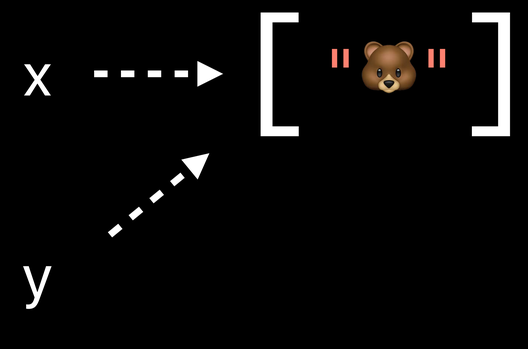


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```

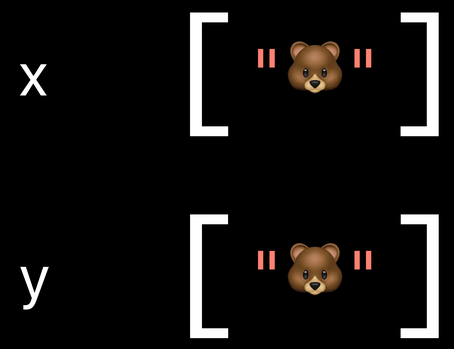



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

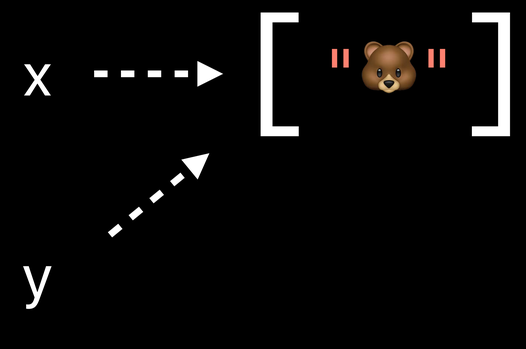


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```

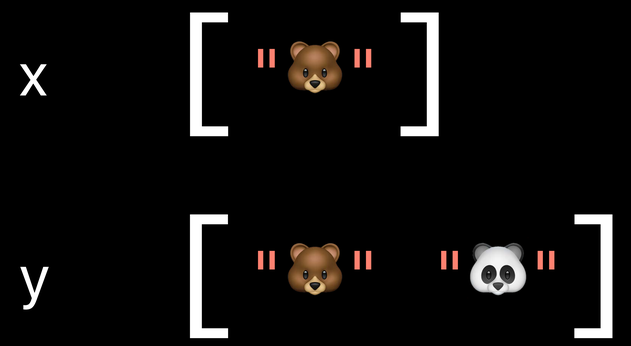



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

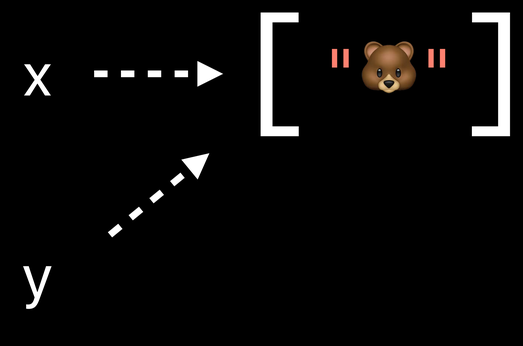


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```

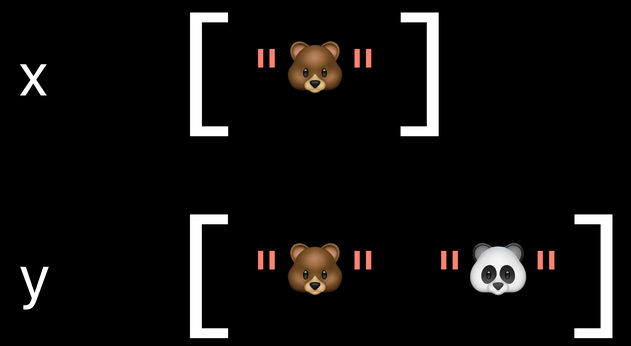



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

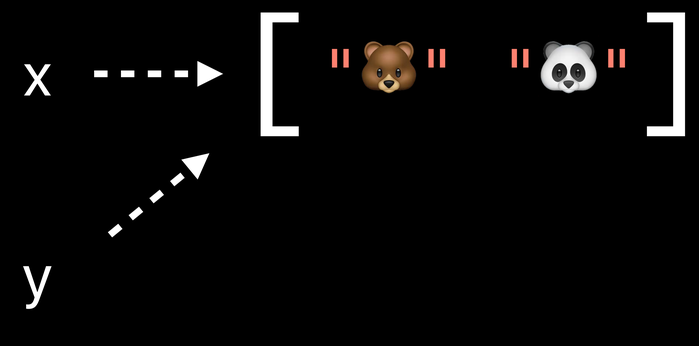


Value and Reference Collections

```
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```




```
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```



Objective-C APIs in Swift

```
// Objective-C
@interface NSView
@property NSArray<NSView *> *subviews;
@end
```

Objective-C APIs in Swift

```objc
// Objective-C
@interface NSView
@property NSArray<NSView *> *subviews;
@end
```

```swift
// Swift
class NSView {
    var subviews: [NSView]
}
```

Bridging

Converts between runtime types

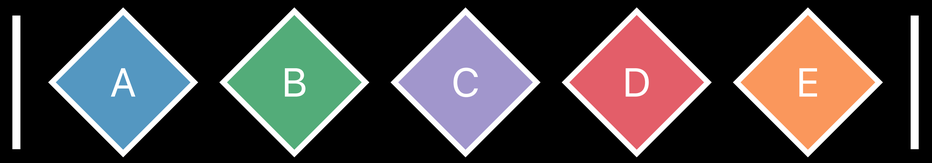
Bidirectional

Bridging of collections

- Is necessary
- Can be cheap, but is never free

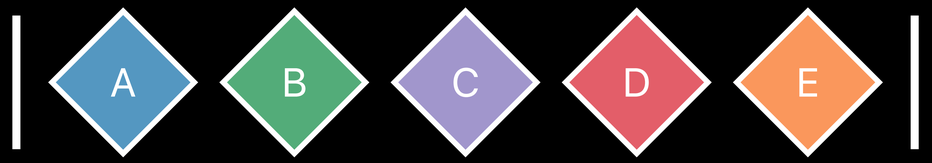
How Bridging Works

Objective-C

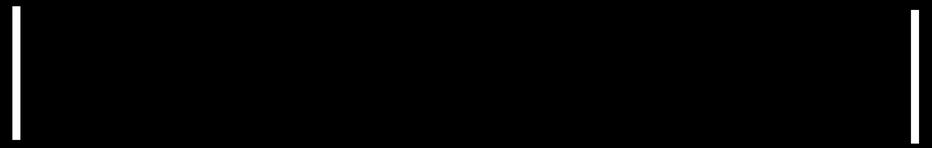


How Bridging Works

Objective-C

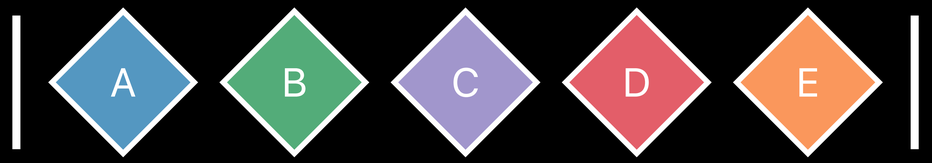


Swift

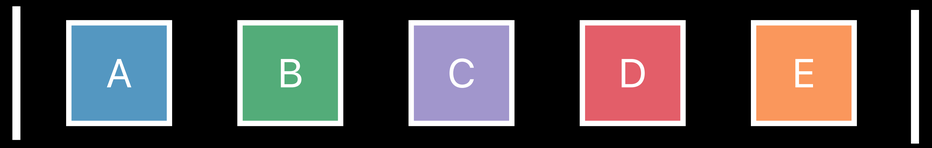


How Bridging Works

Objective-C



Swift



Two Kinds of Bridging

Eager when element types are bridged

Otherwise lazy

- Bridged on first use

Bridging Examples

Bridging Examples

`NSArray<NSData *> *`	`[Data]`	Eager

Bridging Examples

`NSArray<NSData *> *`	`[Data]`	Eager
`NSArray<NSView *> *`	`[NSView]`	Lazy

Bridging Examples

`NSArray<NSData *> *`	`[Data]`	Eager
`NSArray<NSView *> *`	`[NSView]`	Lazy
`NSDictionary<NSString *, id> *`	`[String : Any]`	Eager

Identifying Bridging Problems

Measure your performance with Instruments

Especially inside loops at language boundaries

Look for hotspots like:

- `_unconditionallyBridgeFromObjectiveC`
- `bridgeEverything`

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!    // Range<String.Index>
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)



let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)    // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)



let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

When Bridging Happens

```
text.string
```

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
    var string: String
}
```

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

"Once upon time ... The end."

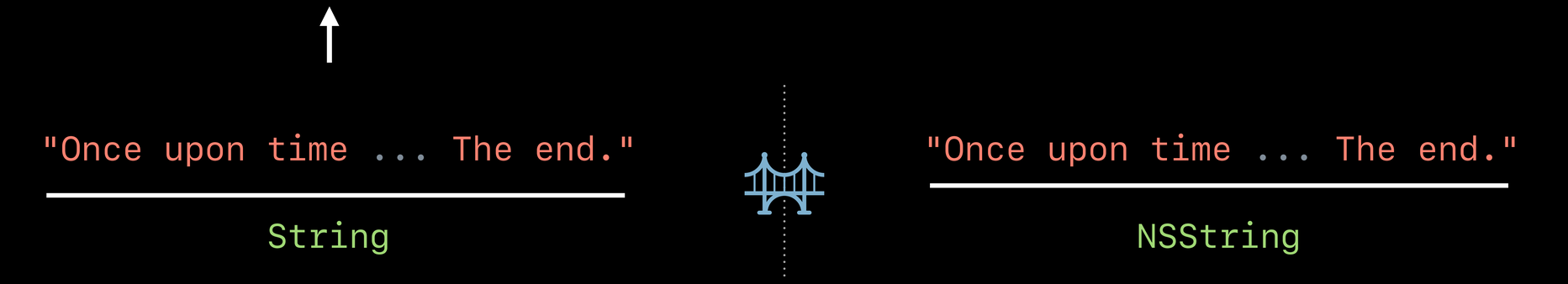
NSString

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```



When Bridging Happens

```objc
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

```objc
@interface NSString : ...
- (NSRange)rangeOfString:(NSString *)string;
@end
```

```swift
class NSMutableAttributedString : ... {
   var string: String
}
```

```swift
struct String {
  func range(of: StringPrococol) -> Range
}
```

When Bridging Happens

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSString : ...
- (NSRange)rangeOfString:(NSString *)string;
@end
```

```
struct String {
  func range(of: StringPrococol) -> Range
}
```

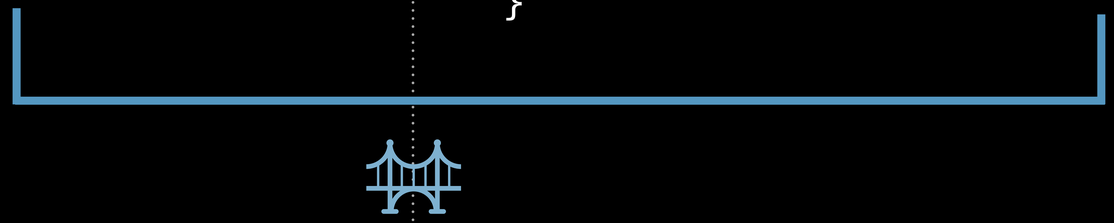
When Bridging Happens

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSString : ...
- (NSRange)rangeOfString:(NSString *)string;
@end
```

```
struct String {
  func range(of: StringPrococol) -> Range
}
```



```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString


let nsrange = string.range(of: "Brown")

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString


let nsrange = string.range(of: "Brown")      // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown")      // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

10 ms

1 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown")    // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

10 ms

1 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown") // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

10 ms

1 ms

3 ms

25 ms

Advice: When to Use Foundation Collections

You need reference semantics

You are working with known proxies

- `NSAttributedString.string`
- Core Data Managed Objects

You've measured and identified bridging costs

Now It’s Your Turn

Explore your existing collections

Measure your code

Audit your mutable state

Gain mastery in Playgrounds

More Information

https://developer.apple.com/wwdc18/229

Cocoa Lab	Technology Lab 7	Friday 11:00AM
Swift Open Hours	Technology Lab 10	Friday 3:00PM


WWDC18